НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО»

Теплоенергетичний факультет

Кафедра автоматизації проектування енергетичних процесів і систем

До захисту допущено:

Завідувач кафедри

_____ Олександр Коваль

«___»_____2020 р.

Дипломна робота

на здобуття ступеня бакалавра
за освітньо-професійною програмою «Програмне забезпечення розподілених систем»
спеціальності 121 «Інженерія програмного забезпечення»
на тему: «Предметно – орієнтоване проектування системи автоматизації споруд»

Виконав:
студент IV курсу, групи ТВ-61
Ташу Альберт Аркадійович _____

Керівник:
Доцент, канд. .ф.-м. наук
Тарнавський Юрій Адамович _____

Рецензент:

_____

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____

Київ - 2020

Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”

Факультет теплоенергетичний

Кафедра автоматизації проектування енергетичних процесів і систем

Рівень вищої освіти перший рівень

Напрям підготовки: 121 Інженерія програмного забезпечення

Спеціалізація: Програмне забезпечення розподілених систем

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Олександр Коваль
(підпис)
”___”_____2020р.

ЗАВДАННЯ

на дипломну роботу студенту

Ташу Альберту Аркадійовичу
(прізвище, ім’я, по батькові)

1. Тема роботи Предметно – орієнтоване проектування системи автоматизації споруд
керівник роботи Тарнавський Юрій Адамович, доцент, кандидат наук
(прізвище, ім’я, по батькові науковий ступінь, вчене звання)

затверджена наказом вищого навчального закладу від ”25” травня 2020р. № 1168-с

2. Строк подання студентом роботи _____

3. Вихідні дані до роботи мова програмування C# 8.0, мова програмування F# 4.6, програмні платформи .NET Core 3.1 та .NET Standard 2.1, модуль об'єктно-реляційного відображення Entity Framework Core, бібліотека MediatR , мова програмування JavaScript, бібліотека React, бібліотека React Material-UI, платформа контейнеризації Docker

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) провести аналіз існуючих систем автоматизації споруд, проаналізувати інструментарій предметно-орієнтованого проектування та можливості його застосування при розробці системи автоматизації споруд, розробити прототип системи керування житловими приміщеннями з

використанням практик предметно-орієнтованого проектування.

5. Перелік ілюстративного матеріалу 1 — Титульний слайд; 2 — Актуальність; 3 — Мета; 4 — Вимоги до прототипу системи; 5 — Предметно-орієнтоване проектування; 6 — Обмежені контексти прототипу системи; 7 — Діаграма класів контексту керування приміщеннями; 8 — Огляд високорівневої структури системи; 9 — Діаграма послідовності додавання вимоги до приміщення; 10 — Технології для розробки системи; 11 — Створення та перегляд приміщень; 12 — Конфігурація приміщення; 13 — Конфігурація приміщення; 14 — Перегляд звітів; 15 — Висновок;

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання ”11” жовтня 2019 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітки
1.	Затвердження теми роботи	08.03 — 14.03	
2.	Вивчення та аналіз задачі	14.04 — 21.04	
3.	Розробка архітектури та загальної структури	21.04 — 27.04	
4.	Розробка структур окремих підсистем	27.04 — 06.05	
5.	Програмна реалізація системи	06.05 — 20.05	
6.	Оформлення пояснювальної записки	20.05 — 08.06	
7.	Захист програмного продукту	08.06	
8.	Передзахист	08.06	
9.	Захист	16.06	

Студент _____ Ташу А. А.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Тарнавський Ю. А.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Розвиток та популяризація технологій, бажання людини шукати нові шляхи для спрощення щоденних проблем та необхідність у комфортному місці проживання — завдяки цим факторам технології автоматизації житлових приміщень у сьогодення мають чудові перспективи до розвитку та поширення.

Існує велика кількість продуктів для автоматизації житлових приміщень, це можуть бути як невеликі прилади укомплектовані власним мобільним додатком для керування станом приладу, так і комплексні системи, що встановлюються до будинку та регулюють величезну кількість параметрів для забезпечення комфортного проживання власників будинку.

Розмаїття автономних недорогих приладів та більш значні матеріальні затрати на встановлення комплексної системи призводиять до потреби у створенні системи для інтеграції різних приладів до єдиної системи автоматизації житлових приміщень.

Метою даної роботи є дослідження доречності застосування практик предметно-орієнтованого проектування для створення системи автоматизації житлових приміщень, здатної вирішити поставлену проблему та розробка програмного прототипу такої системи.

Записка цієї роботи містить 63 сторінок, 26 рисунків, 24 посилання та 3 додатки.

Ключові слова до роботи: DDD, предметно-орієнтоване проектування, розумний дім, інтернет речей, автоматизація споруд, обмежений контекст, єдина мова.

ABSTRACT

The growth and popularization of technology, the desire to find new ways to simplify everyday problems and the need for a comfortable place to live — thanks to these factors, home automation technology today has great prospects for evolution and spread.

There are a large number of products for home automation, it can be small devices equipped with their own mobile application to control the condition of the device, or complex systems that are installed in the house and regulate a huge number of parameters to ensure comfortable living for homeowners.

The variety of autonomous inexpensive devices and the significant material costs for the installation of a comprehensive system leads to the need to create a system for the integration of various devices into a single system for home automation.

The purpose of this work is to explore the appropriateness of the application of domain-driven design practices to create a system for home automation, able to solve the problem and develop a software prototype of such a system.

The note of this work contains 63 pages, 26 images, 24 references and 3 appendices.

Keywords list: DDD, subject-oriented design, smart home, Internet of Things, building automation, limited context, single language.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

1. API (з англ. — Application Programming Interface) — чітко визначений набір контрактів певного програмного модуля з іншими програмними модулями;
2. DDD (з англ. — Domain Driven Design) — набір практик проектування та розробки програмного забезпечення, що направлені на полегшення моделювання предметної області певної системи у програмній реалізації, запропонований Еріком Евансом;
3. DTO (з англ. — Data Transfer Object) — об’єкт, призначений для транспортування даних між підсистемами або програмними шарами;
4. .NET Framework — програмна платформа від компанії Microsoft, складається з CLR, базових бібліотек тощо;
5. CLR (з англ. — Common Language Runtime) — середовище виконання байт-кодів що є результатом компіляції мов програмування, сумісних з .NET Framework.

ВСТУП

Через швидкий розвиток та прогрес у сфері високих технологій значно збільшилась кількість електронних пристроїв, які кожен день використовує сучасна людина. Саме через це під час проектування нового житлового середовища не можливо уникнути необхідності використання технологічних досягнень, які сприяють покращенню життя людини. Сучасна людина має дуже високі вимоги до комфортності середовища проживання. В результаті інженерне оснащення квартир та будинків неухильно ускладняється, і зростає кількість пристроїв, що беруть участь у формуванні цього середовища.

На сьогоднішній день існує велика кількість пристроїв від різноманітних виробників, що дозволяє користувачам автоматизувати певні процеси керування своєю оселею: керування освітленням, система безпеки, система контролю вологості та температури повітря тощо. Зазвичай для окремих пристроїв виробники постачають мобільні додатки або веб-сайти для керування пристроєм та спостереження за певними показниками приміщення в якому даний пристрій оперує. Результатом такого підходу є розмаїття невеликих додатків та веб-сайтів, необхідних користувачу для отримання цілісної картини стану його оселі у певний момент. Для конфігурації бажаних параметрів користувачу також необхідно виконати деякі дії в кожному з наданих додатків або веб-сайтів. У противазі до щойно описаного підходу до автоматизації житлового приміщення існують компанії, що надають комплексні рішення даної проблеми. Зазвичай такі комплексні рішення потребують значних фінансових витрат та обмежуються початковою конфігурацією виробника.

Метою даної роботи є створення системи централізованого керування житловим приміщенням за допомогою інтеграції пристроїв сторонніх

виробників до єдиної платформи, призначеної для автоматизації жилого приміщення.

Для проектування та реалізації програмного комплексу даної роботи було використано практики предметно-орієнтованого проектування, представленого Еріком Евансом у його праці “Domain-Driven Design: Tackling Complexity in the Heart of Software” [1]. Предметно-орієнтоване проектування відмінно підходить для проектування і реалізації програмних систем зі складною моделлю предметної області, що включає в себе велику кількість компонентів, бізнес правил та подій. Прикладом такої предметної області є автоматизація житлового приміщення. До компонентів системи автоматизації житлового приміщення відносяться: модуль керування освітленням, модуль керування вологістю та температурою повітря, модуль системи безпеки, тощо.

1. ЗАДАЧА РОЗРОБКИ ПЛАТФОРМИ ДЛЯ АВТОМАТИЗАЦІЇ ЖИТЛОВИХ ПРИМІЩЕННЬ

Предметно-орієнтоване проектування — набір стратегій та практик проектування та розробки програмного забезпечення, що приділяє основну увагу предметній області системи. Основні принципи предметно-орієнтованого проектування націлені на зменшення рівня складності програмних систем, усунення розбіжностей між термінами предметної області та програмними сутностями у системі.

Платформа для автоматизації житлових приміщень має надавати користувачам змогу авторизації, додавання до свого акаунту житлових приміщень. При подальшому користуванні системою, користувачі мають змогу конфігурувати свої приміщення набором вимірювальних пристроїв для відстеження ряду показників середовища житлового приміщення, таких як: освітлення, температура та вологість повітря. Також користувачі повинні мати змогу реєструвати в системі набір регулюючих пристроїв для зміни стану житлового приміщення: освітлювальні прилади, прилади нагрівання та зволоження повітря. Після конфігурації житлових приміщень користувач повинен мати змогу задати бажані параметри відповідних показників середовища житлового приміщення — бажаний рівень освітлення, температури та вологості повітря у певний проміжок часу.

Результатом даної роботи є прототип системи автоматизації житлових приміщень, спроектований та розроблений у відповідності до принципів так підходів предметно-орієнтованого проектування. При цьому в даній роботі не приділяється увага інтеграції реальних приладів для автоматизації приміщень, прототип має ілюструвати архітектурні підходи до побудови системи

автоматизації житлових приміщень та можливість застосування предметно-орієнтованого проектування для вирішення поставленої проблеми.

1.1 Функціональні задачі поставлені перед системою

До задач, що розв’язує дана система відносяться:

- задача реєстрації користувачів у програмній системі;
- задача конфігурації житлових приміщень вимірювальними та регулювальними приладами користувача;
- задача збору даних вимірювальних приладів відносно окремих приміщень;
- задача керування регулювальними приладами на основі даних отриманих від вимірювальних приладів та бажаних показників середовища житлових приміщень користувача.

1.2 Компоненти програмної системи

Для виконання поставлених перед системою задач передбачається розробка наступних компонентів та підсистем:

- web-інтерфейс користувача;
- вимірювальні пристрої;
- регулювальні пристрої;
- підсистема аутентифікації користувачів та керування акаунтами;
- підсистема конфігурування житлових приміщень користувачів;
- підсистема збору даних вимірювальних пристроїв;
- підсистема керування регулювальними пристроями;

— підсистема аудиту та статистичних показників датчиків та регулювальних пристроїв;

— підсистема моніторингу стану середовища житлових приміщень.

Взаємодію компонентів даної системи зображено на діаграмі компонентів (рисунок 1.1):

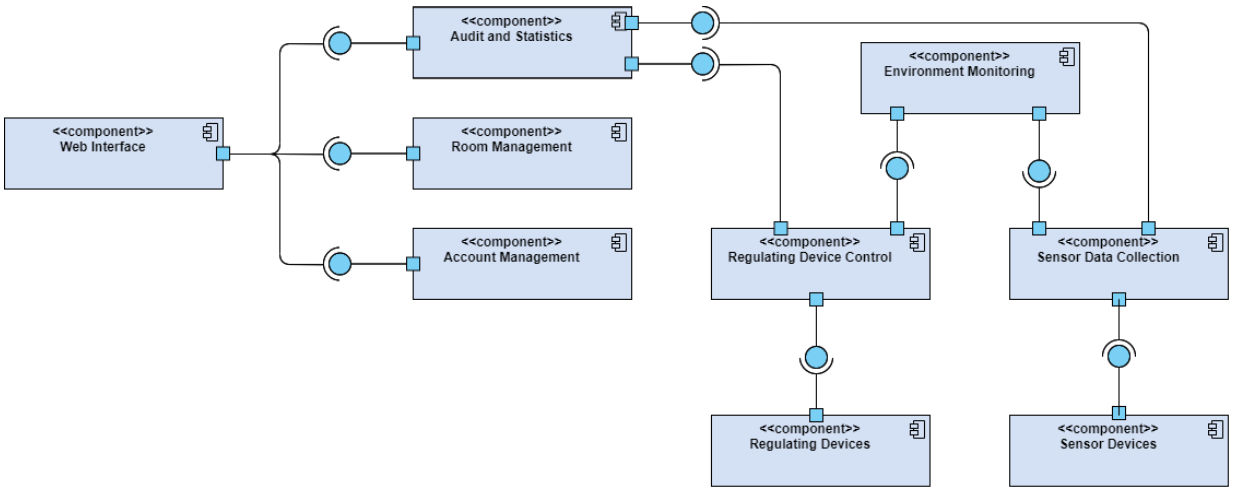

Рисунок 1.1 — Діаграма компонентів системи

Користувач взаємодіє із системою за допомогою Web-інтерфейсу.

Web-інтерфейс взаємодіє із підсистемами аутентифікації та керування акаунтами, конфігурування житлових приміщень для керування власним акаунтом та житловими приміщеннями.

Підсистему аудиту та статистичних показників датчиків та регулювальних пристроїв користувач використовує для отримання даних про періоди активності певних регулюючих пристроїв, перегляд динаміки зміни показників стану середовища житлових приміщень.

Підсистеми збору даних вимірювальних пристроїв та керування регулювальними пристроями виступають у якості адаптерів до певних вимірювальних та регулювальних пристроїв.

До задач підсистеми збору даних ви мірювальних пристроїв відноситься уніфікація повідомлень з вимірювальних пристроїв та передача уніфікованих повідомлень до підсистеми аудиту та статистичних показників датчиків та регулювальних пристроїв та підсистеми моніторингу стану середовища житлових приміщень.

До задач підсистеми керування регулювальними пристроями входить:

— приведення повідомлень, адресованих до певних регулювальних пристроїв, у відповідність до їхнього інтерфейсу;
— повідомлення підсистеми аудиту та статистичних показників про зміну стану певного регулювального пристрою.

До задач підсистеми керування вимірювальними пристроями входить:

— приведення повідомлень, отриманих від певних вимірювальних пристроїв, у відповідальність до загального інтерфейсу системи;
— повідомлення підсистеми аудиту та статистичних показників про отримання даних з вимірювальних пристроїв.

Регулювальні пристрої деяким чином впливають на середовище житлового приміщення — підвищують температуру чи рівень вологості повітря, збільшують рівень освітлення тощо. Вони отримують команди від підсистеми керування регулювальними пристроями у вигляді повідомлень, що реалізують інтерфейс певного пристрою.

До задач підсистеми моніторингу стану середовища житлових приміщень входить аналіз даних, отриманих від підсистеми збору даних вимірювальних пристроїв, перевірка дотримання показників стану середовища житлових приміщень у границях, встановлених користувачем для певного житлового приміщення та надсилання команд до підсистеми керування регулювальними пристроями для зміни стану середовища житлового приміщення у разі необхідності.

1.3 Потенційні користувачі

Потенційні користувачі системи, розробленої під час виконання даної роботи — це власники житла, що мають бажання автоматизувати процеси керування житловим приміщенням. Система орієнтована на більш дешевий сегмент ринку ніж комплексні рішення по встановленню систем автоматизації житлових приміщень, потенційні користувачі — люди що мають бажання спробувати модульно автоматизувати оселю, використовуючи ту кількість приладів що є в наявності із можливістю в подальшому розширити інфраструктуру своєї оселі за рахунок інтеграції нових пристроїв до системи.

2. ВИКОРИСТАННЯ ПРЕДМЕТНО-ОРІЄНТОВАННОГО ПРОЕКТУВАННЯ ПРИ РОЗРОБЦІ СИСТЕМИ АВТОМАТИЗАЦІЇ СПОРУД

2.1 Системи автоматизації житлових приміщень

Системи автоматизації житлових приміщень зазвичай називають системами створення розумного дому або розумного будинку.

Розумний дім — це середовище для проживання, вдосконалене автоматичними системами. Розумний дім здається "розумним", оскільки його щоденна діяльність контролюється комп’ютером. Розумний дім складається з багатьох технологій що слугують для покращення якості життя за допомогою домашніх мереж. Розумний дім — це місце, оснащене високорозвиненими автоматичними системи управління та моніторингу — системами керування освітленням, температурою, різноманітними приладами, мультимедійним обладнанням, системами охорони та багатьма іншими функціями [1].

На сьогоднішній день розробка системи управління розумним будинком є одним з пріоритетних напрямів розвитку автоматизованих систем. У сучасній системі розумного будинку, кожна з систем працює в оптимальному режимі за рахунок взаємообміну даними з іншими системами будівлі, що в підсумку дозволяє максимізувати ефективність роботи всієї системи в цілому. Існують наступні методи управління розумним будинком:

— автоматичне керування розумним будинком на основі датчиків;
— управління розумним будинком за допомогою пульта дистанційного управління і панелі керування;
— віддалене керування.

Система керування будинком на основі датчиків і мікроконтролерів використовується для керування такими системами, як: системи клімат контролю, системи безпеки, системи контролю освітлення, системи контролю доступу тощо. В системі містяться наступні типи датчиків: датчик присутності, датчик диму, датчик руху, датчик температури, датчик освітлення, датчик протікання води тощо [2].

Розвиток систем автоматизації споруд тісно пов’язаний із технологіями інтернету речей.

Інтернет речей, як випливає з назви, — це зв’язок повсякденних пристроїв між собою. З розвитком технологій численні пристрої використовують датчики, приводи, вбудовані комп’ютерні системи та хмарні обчислення. Це дозволило пристроям спілкуватися між собою. Простіше кажучи, Інтернет речей дає можливість пристроям (речам) взаємодіяти та координувати один з одним, тим самим зменшуючи втручання людини в основні повсякденні завдання [3].

2.2 Предметно-орієнтоване проектування

Темою даної роботи є використання предметно-орієнтованого проектування при побудові системи автоматизації споруд.

Термін “Предметно-орієнтоване проектування” було започатковано Еріком Евансом у книзі “Domain-Driven Design: Tackling Complexity in the Heart of Software” [4].

В процесі розробки програмного забезпечення вистачає всіляких труднощів. Головне — це природна складність предметної області, до якої належить задача що потребує вирішення. Уникнути цієї складності неможливо — її можна тільки стримувати. Для контролю складності програмного забезпечення розробляються різні парадигми програмування, мови, підходи та

шаблони проектування. Основною метою предметно-орієнтованого проектування також є уникнення занадто швидкого росту складності програмних систем [4].

Предметно-орієнтоване проектування — це і образ мислення, і система пріоритетів, покликана прискорити розробку програмних продуктів, які застосовуються в складних областях діяльності [4].

Предметно-орієнтоване проектування приділяє багато уваги до моделі предметної області системи.

Модель — це спрощення, така інтерпретація реальності, при якій з явища витягуються істотні для вирішення завдання аспекти, а зайві деталі ігноруються. Модель являє собою узгоджений між розробниками спосіб структуризації знань з предметної області, а також виділення елементів, що представляють найбільший інтерес [4].

Предметно-орієнтоване проектування передбачає використання технік “стратегічного” та “тактичного” проектування [5].

До технік стратегічного проектування належать:

— єдина мова (Ubiquitous language);
— обмежений контекст (Bounded context).

До технік тактичного проектування належать:

— сутність (Entiti);
— об’єкт-значення (Value Object);
— агрегат (Aggregate);
— сховище (Repository);
— фабрика (Factory);
— модуль (Module);
— сервіс предметної області (Domain service);
— подія предметної області (Domain event).

2.2.1 Єдина мова

Проект стикається з серйозними проблемами, коли в ньому відсутня єдина мова. У фахівців предметної області є свій жаргон, а у розробників — власні назви програмних сутностей, явищ та компонентів, пристосовані для опису предметної області в термінах програмної архітектури. Необхідність в перекладі ускладнює комунікацію і послаблює інтенсивність використання існуючих знань [4].

Ця мова повинна ґрунтуватися на моделі предметної області, що використовується в програмному забезпеченні — отже, вона має бути конкретною та однозначною, оскільки програмне забезпечення не справляється із неоднозначністю [6].

Вдала модель, будь вона велика чи маленька, не повинна мати логічних невідповідностей визначень що перекриваються. Корпоративні системи зазвичай інтегрують велику кількість підсистем або містять компоненти настільки різні, що мало які елементи предметної області сприймаються ними односторонньо. Вимога уніфікувати модель в усіх частинах системи надмірна і нездійсненна. Але розробник все ж має можливість уникнути подальшого виродження моделі. Для цього потрібно явно визначити обмежений контекст, всередині якого може бути застосована модель, а потім, при необхідності, визначити його взаємозв'язок з іншими обмеженими контекстами [4].

2.2.2 Обмежений контекст

Обмежений контекст можна розглядати в якості простору імен для єдиної мови. Явища та сутності предметної області можуть мати однакову назву у деяких обмежених контекстах, але за однаковими назвами можуть стояти різні семантичні значення. В якості прикладу можна навести сутність “Користувач” для контексту, відповідального за замовлення та доставку товарів, та сутність “Користувач” для контексту платіжної системи. В обох

контекстах назва сутності співпадає, але при цьому кожний із контекстів передбачає різні властивості та атрибути сутності [4].

Обмежений контекст — це явна межа, в межах якої існує модель предметної області. Всередині межі всі терміни, назви та фрази що є частиною Єдиної мови мають специфічне значення, і модель точно відображає цю мову [5].

При використанні предметно-орієнтованого проектування, виявлення обмежених контекстів — один із найважливіших етапів проектування. Обмежені контексти можуть впливати на вибір кількості та типів баз даних, на побудову окремих компонентів системи у вигляді сервісів що працюють в окремому процесі, розподілення розробників на команди тощо.

Обмежений контекст — це також і ключове поняття в багатьох сервіс-орієнтованих архітектурах, зокрема в мікросервісній архітектурі.

Для визначення взаємодії між обмеженими контекстами у системі використовуються карти контекстів, що перелічують всі існуючі у системі обмежені контексти та способи їх взаємодії між собою.

2.2.3 Сутність

Сутність — це поняття програмного забезпечення, для котрого обов’язковим показником є унікальність [7]. Такі об’єкти зазвичай існують протягом довгого періоду часу та змінюють свій стан впродовж часу свого існування. Прикладами сутностей у системі автоматизації житлових приміщень є: Користувач, Приміщення, Конфігурація бажаного стану приміщення тощо.

2.2.4 Об’єкт-значення

Об’єкти-значення — це одні із ключових будівельних блоків в системах, побудованих із використанням предметно-орієнтованого проектування. На

відміну від сутностей, об’єкти-значення не мають вимоги бути унікальними. Навпаки — при роботі з об’єктами-значеннями використовується перевірка структурної рівності для виявлення еквівалентних об’єктів. Структурна рівність передбачає що два об’єкти є рівними, якщо рівні їх відповідні атрибути. Окрім того, об’єкти-значення зазвичай незмінні після створення [5].

Прикладами об'єктів-значень у програмних системах є класи для представлення дати, часу, строкових значень, номерів телефонів, кількості грошей у певній валюті тощо. Обширне використання об'єктів-значень призводить до спрощення програмної системи в цілому, адже замість того щоб використовувати узагальнені типи даних для представлення значень, що насправді належать до різних типів даних, використовуються більш конкретні типи даних. Це призводить до виявлення більшої кількості помилок ще на етапі компіляції програмних кодів [8].

2.2.5 Агрегат

Агрегат — це сукупність сутностей та об'єктів-значень, згрупованих разом для спрощення роботи у комплексних сценаріях. Агрегати використовуються для спрощення виконання трансакційних операцій, для зменшення рівня зв’язаності системи та для спрощення підтримки інваріантів. Інваріанти — це контрольне твердження щодо агрегату, яке повинно бути істинним завжди, крім специфічних перехідних станів, таких як процес виконання методу або незавершена транзакція бази даних. У кожного агрегату є корінь (Aggregate Root) та границя. Границя агрегату обмежує сукупність сутностей та об’єктів-значень що входять до агрегату від інших частин системи. Корінь агрегату — це сутність, яка виступає у ролі точки доступу до всього агрегату цілком ззовні його границь. Будь-який компонент програмної системи може отримати доступ до агрегату тільки через його корінь. Такі обмеження зменшують кількість зв’язків між компонентами програмної

системи, полегшують роботу із зовнішнім сховищем даних, транзакціями тощо.

2.2.6 Сховище

Сховище представляє всі об'єкти певного типу у вигляді концептуальної множини, зазвичай віртуальної, емульованої. Воно працює аналогічно колекції, тільки з більш розвиненим механізмом запитів. Можна додавати і видаляти об'єкти відповідного типу, приховані механізми сховища будуть автоматично розміщувати їх у базі даних або видаляти з неї. Ввівши це визначення, отримуємо повний набір засобів для доступу до кореневих об'єктів агрегатів з самого початку і до кінця їх циклу існування. Клієнти запитують об'єкти зі сховища, використовуючи методи запитів, які відбирають об'єкти за критеріями, що задає клієнт, — зазвичай по значенням певних атрибутів. Сховище видає запитуваний об'єкт або множину об’єктів та інкапсулює механізми запитів до бази даних, фільтрації та відображення метаданих. Сховища можуть реалізувати безліч найрізноманітніших запитів, відбираючи об'єкти в залежності від встановлених клієнтами критеріїв. Вони також можуть повертати інформацію зведеного характеру, — наприклад, скільки примірників об'єктів задовольняють ті чи інші критерії. Сховище може навіть виконувати обчислення за підсумками запиту. Зазвичай сховища використовуються для роботи з базами даних, але це не обов’язковий критерій. Сховища можуть інкапсулювати роботу з будь-яким джерелом даних [4].

2.2.7 Фабрика

Фабрики використовуються для побудови складних об’єктів або сукупностей об’єктів. Створення об'єкта саме по собі може бути дуже важливою операцією, але збірка об'єкта — це абсолютно не та робота, яку потім доведеться робити цьому ж об'єкту в ході свого існування. Об'єднання

цих обов'язків в одному об'єкті породжує незграбні, важкі для розуміння програмні конструкції. Якщо дати клієнту можливість керувати створенням об'єктів, це спотворює архітектуру самого клієнта, порушує інкапсуляцію зібраного об'єкта або агрегату, а також занадто сильно прив'язує клієнта до конкретної реалізації створюваного ним об'єкта. Створення складних об'єктів — це обов'язок рівня предметної області, але не об'єктів, які безпосередньо виражають модель [4]. Фабрики зазвичай реалізують один із породжувальних шаблонів проектування: шаблонний метод, абстрактну фабрику, будівельник, прототип або сінглтон [9].

2.2.8 Модуль

Модулі використовуються для агрегації пов'язаних об'єктів предметної області та відокремлені від об'єктів, які не є пов'язаними або є слабо пов'язаними. Обмежені контексти часто охоплюють кілька модулів. У деяких мовах програмування існують поняття аналогічні до модулів у предметно-орієнтованому проектуванні — пакети або простори імен [5].

2.2.9 Сервіс предметної області

Сервіс — це операція, наявна в моделі у вигляді відокремленого інтерфейсу, який, на відміну від сутностей і об’єктів-значень, не інкапсулює ніякого стану. Сервіс — це звичайне поняття в багатьох програмних системах, але сервіси можуть бути застосовані і на рівні предметної області. На відміну від сутностей і об’єктів-значень, сервіси визначаються тільки тим, що вони вміють робити для клієнта. Відповідно, і ім'я сервісу зазвичай дають по його функції, а не суті - це дієслово, а не іменник. Визначення у нього, проте, може бути досить абстрактним, але з іншим відтінком в порівнянні з об'єктом. На сервіс повинні покладатися конкретні обов'язки, і вони разом з інтерфейсом повинні визначатися в складі моделі. Імена операцій слід взяти з єдиної мови.

Параметрами і результатами роботи сервісу повинні бути об'єкти моделі. Не варто, однак, і зловживати сервісами, позбавляючи сутності та об’єкти-значення всякого функціонального навантаження [4].

Правильно реалізований сервіс володіє трьома властивостями:

— виконувана сервісом операція відповідає поняттю моделі, яке не є частиною сутності або об'єкта-значення;
— інтерфейс сервісу визначено через інші елементи моделі предметної області;
— сервіс не має власного стану [4].

2.2.10 Події предметної області

Події предметної області використовуються для того, щоб зафіксувати виникнення певного явища, що сталося у предметній області. Це надзвичайно потужний інструмент моделювання. Подія предметної області — це повноцінна частина моделі предметної області, відображення того, що трапилося у предметній області. Події породжуються агрегатами. Після цього вони потрапляють до компоненту публікації подій, котрий розповсюджує інформацію про виникнення певної події до набору обробників подій. Події, що утворились під час виконання певної транзакції над агрегатом можуть виконуватись як і у межах тієї самої транзакції, так і в контексті окремих транзакцій. Використання подій предметної області дозволяє більш точно відобразити предметну область у моделі та спростити код програмних компонентів шляхом відв’язування моменту появлення певного явища у програмній системі від обробки цього явища. За рахунок розділення обов’язків ініціювання події та її обробки знижується рівень зв’язаності системи та полегшується подальша підтримка коду системи.

2.3 Аналіз існуючих аналогічних програмних систем

Задачею даної роботи є створення цілком програмної системи автоматизації житлових приміщень, використовуючи практики предметно-орієнтованого проектування, з метою інтеграції сторонніх апаратних пристроїв. З огляду на цей факт, досить повних аналогів не було знайдено. Більшість систем для автоматизації житлових приміщень є комбінацією програмних та апаратних засобів та пропонують автономне рішення більш або менш повного спектру проблем автоматизації приміщень. Далі буде розглянуто деякі з таких систем.

2.3.1 Система Xiaomi Smart Home

Xiaomi — це китайська компанія, утворена в 2010 році. Компанія стала відома завдяки своїй прошивці MIUI, на базі Android і фірмовим смартфонам. В кінці 2014 року компанія Xiaomi представила систему автоматизації житлових приміщень, що складається з розумної розетки (Mi Smart Power Plug), камери спостереження (Yi Camera), розумної лампочки (Yeelight LED) і блоку управління побутовою технікою (IR Remote Controller). З того часу компанія розвиває продукт та періодично випускає нові пристрої до існуючої системи.

Переваги системи Xiaomi Smart Home:

— пристрої, які входять до цієї системи можна придбати окремо і вони працюють автономно;
— простий спосіб підключення нових пристроїв за допомогою смартфону;
— невелика ціна порівняно з іншими системами даного сегменту.

Недоліки системи Xiaomi Smart Home:

— непроробленa інтернаціоналізація та переклад з китайської мови.

2.3.2 Система Fibaro Home Center

Fibaro — це бездротова інтелектуальна система автоматизації споруд, що працює на протоколі Z-Wave. До стартового набору автоматизації Fibaro входять:

— контролер Home Center, до обов'язків котрого входить керування системою;
— датчик протікання;
— датчик диму;
— датчик відкриття та закриття дверей;
— розумна виделка.

Переваги системи Fibaro Home Center:

— на основі комплекту можна створити професіональний розумний дім, який охоплює велику кількість процесів роботи електричних приладів;
— наявна система запобігання надзвичайних ситуацій у вигляді датчиків диму та протікання.

Недоліки системи Fibaro Home Center:

— потрібна центральна система, яка управляє всіма пристроями, окремо вони не працюють;
— для встановлення системи обов'язково потрібно встановити додаток з офіційного сайту, система встановлюэться локально.

2.3.3 Система автоматизації житлових приміщень Brilliant Smart Home

Brilliant — це система освітлення та управління, яка дозволяє власникам будинків легко керувати освітленням, дзвінками, замками, камерами, музикою, кліматом, домофоном тощо. Систему легко встановити, інтегрувати з провідними брендами у кожну категорію розумного дому та перетворити будь-який будинок чи квартиру в простий у користуванні розумний будинок.

Система встановлюється на місце звичайного вимикача світла та може керувати освітленням приміщення. Окрім цього Briliant Smart Home має ряд інтеграцій з приладами для автоматизації оселі від сторонніх виробників та інтеграцію з Amazon Alexa для керування голосом.

Переваги системи Briliant Smart Home:

— можливість керувати системою за допомогою смартфону або настінного дисплею;
— наявність інтеграції з приладами для автоматизації житлових приміщень сторонніх виробників;
— наявність інтеграції з Amazon Alexa.

Недоліки системи Briliant Smart Home:

— необхідність встановлення настінного контролера;
— невелика кількість пристроїв для інтеграції.

3. ЗАСОБИ РОЗРОБКИ ПРОГРАМНОГО ПРОДУКТУ

Сучасні технології для розробки програмного забезпечення допомагають пришвидшити кожний з етапів розробки програмного продукту. В даному розділі приведено опис мов програмування, фреймворків, бібліотек та платформ, що були використані під час розробки прототипу системи автоматизації споруд.

3.1 Мова програмування C#

C # — сучасна об'єктно-орієнтований і безпечна по відношенню до типів мова програмування. C # відноситься до широко відомого сімейства мов C [10].

C # є об'єктно-орієнтованою мовою, але підтримує також і компонентно-орієнтоване програмування. Розробка сучасних додатків все більше тяжіє до створення програмних компонентів у формі автономних і само документованих пакетів, що реалізують окремі функціональні можливості. Головна особливість таких компонентів в тому, що вони являють собою модель програмування з властивостями, методами і подіями. У них є атрибути, що надають декларативні відомості про компоненти. Вони включають в себе власну документацію. C# надає мовні конструкції, що безпосередньо підтримують таку концепцію роботи. Завдяки цьому C # підходить для створення і застосування програмних компонентів [10].

При розробці даної системи було використано мову програмування C# версії 8.0. Останні оновлення принесли до мови багато можливостей із функціональних мов програмування, що надають можливості до побудови

більш прозорих, надійних та придатних до модульного тестування програмних компонентів.

3.2 Мова програмування F#

F# — функціональна мова програмування для платформи .NET від компанії Microsoft із відкритим вихідним кодом. Мова характеризується легким синтаксисом, незмінністю за замовчуванням, потужними засобами виведення типів та багатим інструментарієм для створення типів даних [11].

Мова F# сумісна з іншими мовами платформи .NET, що дозволяє використовувати сильні сторони даної мови навіть якщо більша частина програмного коду написана із використанням іншої мови програмування на платформі .NET [11].

При розробці системи автоматизації житлових приміщень, мова F# використовувалась в якості зручного та надійного інструменту для створення типів для незмінних об’єктів-повідомлень що використовуються при комунікації між компонентами системи.

3.3 Платформа .NET Core

.NET Core — це платформа розробки загального призначення з відкритим кодом, призначена для створення кросплатформних додатків. Платформа дозволяє створювати додатки для Windows, macOS і Linux з підтримкою процесорів x64, x86, ARM32 і ARM64. Платформа надає доступ до інтерфейсів для створення хмарних додатків, додатків для Інтернету речей, використання клієнтського інтерфейсу і машинного навчання [12].

Для .NET Core притаманні наступні характеристики [12]:

— кросплатформність — підтримка операційних систем Windows, macOS і Linux;
— відкритий код — платформа .NET Core базується на відкритому коді і поширюється за ліцензіями MIT і Apache 2. .NET Core є проектом .NET Foundation;
— сучасні технології — у платформі реалізовані сучасні парадигми, такі як асинхронне програмування, шаблони без копіювання з використанням структур та управління ресурсами для контейнерів;
— продуктивність — платформа забезпечує високу продуктивність за рахунок таких можливостей, як вбудовані апаратні компоненти, багаторівнева компіляція тощо;
— узгодженість між середовищами — однакове виконання коду в різних операційних системах і архітектурі, включаючи x64, x86 і ARM;
— інтерфейс командного рядка — зручні засоби командного рядка для локальної розробки та безперервної інтеграції;
— гнучка розробка — .NET Core можна включити в додаток або встановити паралельно (на рівні користувача або системи). Можливість використання з контейнерами Docker.

3.4 Реалізація публічного Web API на базі ASP.NET Core

Під час проектування програмного продукту було прийнято рішення реалізувати для програмної системи Web API. На прийняття цього рішення вплинули наступні фактори:

— перспектива створення множини клієнтських додатків на різних платформах із використанням єдиної реалізації основної логіки програмної системи;

— розділення задач реалізації інтерфейсу користувача та реалізації логіки програмної системи.

Web API для системи автоматизації житлових приміщень було реалізовано за допомогою ASP.NET Core.

ASP.NET Core — це кросплатформна, високопродуктивна система з відкритим вихідним кодом для створення сучасних додатків, підключених до Інтернету, із можливістю інтеграції з хмаринними сервісами [13].

3.5 Реалізація бази даних та технології доступу до даних

При розробці системи автоматизації житлових приміщень було прийнято рішення використовувати Microsoft SQL Server 2019 у якості системи управління базами даних. В якості технології доступу до даних із коду програмної системи було обрано Entity Framework Core.

Microsoft SQL Server 2019 — це система управління реляційними базами даних, розроблена Microsoft. До основних задач даної системи управління базами даних належить забезпечення можливості сторонніх застосунків зберігати дані та отримувати дані за допомогою запитів на мові Transact-SQL.

Entity Framework Core — це модуль об’єктно-реляційного відображення, що дозволяє розробникам .NET працювати з базою даних за допомогою .NET-об’єктів та виключати потребу в більшості кодів доступу до даних. Entity Framework Core надає можливість здійснювати доступ до даних за допомогою моделі. Модель складається з класів сутностей та об'єкта контексту, який представляє сеанс із базою даних, що дозволяє запитувати та зберігати дані [14].

3.6 Реалізація комунікації між компонентами програмної системи

При розробці програмної системи було поставлено наступні задачі інтеграції програмних компонентів, що виконуються у різних процесах:

— необхідно забезпечити засіб інтеграції програмних компонентів, реалізуючи подієву модель для забезпечення розповсюдження подій предметної області, що є змістовними для декількох обмежених контекстів, код яких виконується в окремих процесах;

— необхідно забезпечити інтеграцію Web-інтерфейсу з публічним Web API.

Для вирішення поставлених задач було обрано наступні технології:

— реалізацію протоколу AMPQ брокером повідомлень RabbitMQ для забезпечення подієвої моделі інтеграції;

— протокол HTTP для комунікації між Web-інтерфейсом та публічним Web API.

3.6.1 Протокол AMPQ та брокер повідомлень RabbitMQ

AMPQ (Advanced Message Queuing Protocol) — це протокол обміну бінарними повідомленнями, який дає змогу клієнтським програмам спілкуватися за допомогою брокерів повідомлень. Протокол використовується для комунікації мікросервісів та корпоративних програмних систем. Брокери обміну повідомленнями отримують повідомлення від видавників (додатків, які публікують повідомлення) і направляють їх до споживачів (додатків, які обробляють повідомлення) [15].

RabbitMQ — це брокер повідомлень, що реалізує протокол AMPQ.

Нижче неведено схему розповсюдження повідомлення від видавника до споживача за допомогою RabbitMQ (рисунок 3.1):

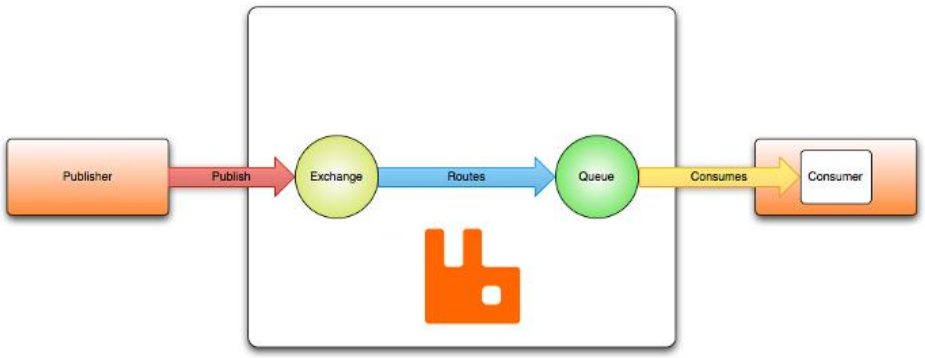

Рисунок 3.1 — Схема розповсюдження повідомлення від видавника до споживача за допомогою RabbitMQ

Exchange — це агрегатори повідомлень, об'єкти AMQP, куди надсилаються повідомлення. Агрегатори приймають повідомлення і направляють його в нуль чи більше черг. Використовуваний алгоритм маршрутизації залежить від типу агрегатору та правил маршрутизації [15].

Queue — це черги, в моделі AMQP вони зберігають повідомлення, які споживаються програмами-споживачами [15].

3.6.2 Протокол HTTP

HTTP (Hyper Text Transfer Protocol) — протокол рівня додатків для розподілених, інформаційних систем гіпермедіа. Це загальний протокол без зберігання стану між запитами, який може бути використаний для багатьох завдань поза його використанням для гіпертексту, таких як розподілені системи управління об'єктами, шляхом розширення методів запиту, кодів помилок та заголовків [16].

3.7 Реалізація Web-інтерфейсу користувача

При розробці Web-інтерфейсу користувача було поставлено наступні задачі:

— побудова візуального інтерфейсу користувача;
— побудова графіків для зображення значень показників вимірювальних приладів у житловому приміщенні;
— комунікація з публічним Web API за допомогою HTTP.

3.7.1 Побудова візуального інтерфейсу користувача

Для побудови візуального інтерфейсу користувача були використані такі технології, як HTML, CSS, JavaScript, React, React Material-UI.

Мова розмітки HTML є стандартним засобом для побудови Web-інтерфейсів. Використання HTML у купі із таблицями стилів CSS надає широкі можливості для створення Web-сторінок з варіативним дизайном на смак розробника.

Мова програмування JavaScript використовується для надання інтерактивності Web-сторінкам.

JavaScript — це інтерпретована, нетипізована мова програмування з об'єктно-орієнтованими можливостями. З точки зору синтаксису базова мова JavaScript нагадує C, C ++ і Java. В основі мови JavaScript і підтримуваних ним типів даних лежать міжнародні стандарти, завдяки чому забезпечується прекрасна сумісність між реалізаціями [17].

З метою спрощення побудови веб-інтерфейсу користувача було прийнято рішення використовувати бібліотеку React.

React — це бібліотека JavaScript для створення інтерфейсів користувача. React використовує декларативний підхід до створення інтерфейсу користувача. Інтерфейс будується із компонентів, що представляють собою сукупність візуальних характеристик та певних даних - стану. При зміні стану, бібліотека React оновлює візуальне представлення компоненту самостійно, не

потребуючи додаткових дій від розробника. Для написання компонентів використовується мова JavaScript [18].

Для уніфікації зовнішнього вигляду веб-інтерфейсу користувача було використано бібліотеку React компонентів — React Material-UI. Бібліотека надає набір ізольованих компонентів для побудови інтерфейсів з уніфікованим стилем оформлення.

3.7.2 Побудова графіків

Для побудови графіків було обрано бібліотеку plotly.js — це декларативна бібліотека графіків високого рівня. Plotly.js постачається з більш ніж 40 типами діаграм, включаючи 3D-діаграми, статистичні графіки та SVG-карти. plotly.js є безкоштовною бібліотекою з відкритим кодом [19].

3.7.3 Комунікація з публічним Web API за допомогою HTTP

Для вирішення задачі комунікації з публічним Web API за допомогою HTTP протоколу було обрано бібліотеку axios.

Axios — це бібліотека з відкритим кодом, що надає інтерфейс для користування асинхронним HTTP клієнтом у браузері або у середовищі node.js.

3.8 Організація покриття функціоналу системи модульними тестами

Модульне тестування – важлива частина розробки програмного забезпечення. Наявність тестів дає розробникам змогу легше проводити рефакторинг коду, раніше виявляти внесені до логіки програмної системи помилки та загалом сприяє написанню менш зв'язанного коду.

В якості фреймворку для модульного тестування було обрано NUnit — це фреймворк для модульного тестування на платформі .NET з відкритим кодом. NUnit дозволяє створювати та виконувати набори модульних тестів та підтримує зчитування вхідних даних для тестів з стороннього джерела даних.

3.9 Інтегроване середовище розробки Visual Studio 2019

Інтегроване середовище розробки Visual Studio — це програмний продукт, який може бути використаний для редагування, налагодження та написання коду, а в подальшому — публікації додатку. Інтегроване середовище розробки (IDE) — це багатофункціональна програма, яка може бути використана для багатьох аспектів розробки програмного забезпечення. Окрім стандартного редактора та відладчика, який надає більшість IDE, Visual Studio включає компілятори, інструменти для доповнення коду, графічні дизайнери та багато інших функцій для полегшення процесу розробки програмного забезпечення.

3.10 Розгортання та виконання коду системи

3.10.1 Система управління контейнерами Docker

В якості платформи для виконання коду програмної системи було обрано систему управління контейнерами Docker.

Docker — це відкрита платформа для розробки, доставки та запуску програм. Docker дозволяє відокремити програми від інфраструктури, що дозволяє швидко розгортати програмне забезпечення. Використання Docker для розробки, тестування та швидкого розгортання коду надає можливість значно зменшити затримку між написанням коду та його запуском у виробництві [20].

Docker забезпечує можливість упаковки та запуску програми у слабко ізольованому середовищі, званому контейнером. Ізоляція дозволяє одночасно запускати багато контейнерів на одному хості. Контейнери потребують небагато ресурсів хоста, тому що їм не потрібно додаткового навантаження у вигляді гіпервізора, як, наприклад, віртуальним машинам. Контейнери працюють безпосередньо в ядрі хост-машини. Це означає, що можливо запустити більше контейнерів на певній комбінації апаратних засобів, ніж при використанні віртуальних машин. Контейнери Docker можна запускати навіть в хост-машинах, які фактично є віртуальними машинами [20].

Docker керує контейнерами, котрі він створює на основі образів. Docker образ — це шаблон, доступний лише для читання, з інструкціями щодо створення контейнера Docker. Часто образ базується на іншому образі, з деякими додатковими налаштуваннями. Можна використовувати існуючі образи або створювати власні за допомогою опису команд для створення образу у Dockerfile. Контейнер — це запущений екземпляр образу. Контейнер можна створити, запустити, зупинити, перемістити або видалити за допомогою Docker. Контейнер можна підключити до однієї або декількох мереж, приєднати до нього сховище або навіть створити новий образ залежно від поточного стану певного існуючого контейнера. Контейнер визначається його зображенням, а також будь-якими параметрами конфігурації, наданими йому під час створення або запуску. Коли контейнер припиняє існування, будь-які зміни його стану, які не зберігаються в постійному сховищі, зникають [20].

3.10.2 Інструмент управління багатоконтейнерними застосунками Docker Compose

Docker Compose — це інструмент для визначення та запуску багатоконтейнерних програм. Compose надає можливість налаштувати сервіси

певної програмної системи у файлі конфігурації формату YAML. Налаштування Docker Compose дозволяє описати середовище виконання програми в одному файлі та в подальшому використовувати ці налаштування при розгортанні програми у будь-якому середовищі. Після створення конфігураційного файлу Docker Compose може бути використаний для запуску всією системи контейнеризованих програм разом із певними налаштуваннями середовища їх виконання.

3.11 Засоби контролю процесу розробки системи

3.11.1 Система контролю версій Git

Git — це розподілена система управління версіями, яка дозволяє відстежувати зміни у файлах [21].

Розподілені системи управління версіями — це системи контролю версій, у котрих клієнти не просто зберігають останні версії файлів, вони повністю відображають репозиторій, включаючи його повну історію. Таким чином, якщо будь-який сервер вийде з ладу, будь-який клієнтський репозиторій може бути скопійовано назад на сервер, щоб відновити його. Кожен клон - це справді повна копія всіх даних [21].

Особливість Git, яка насправді робить його відокремленим від майже всіх інших систем контролю версій, це його модель розгалуження. Git дозволяє мати кілька локальних гілок, які можна представити як напрямки розвитку стану підконтрольної директорії та можуть бути повністю незалежними один від одної. Створення, об'єднання та видалення цих напрямків розвитку займає секунди.

3.11.2 Віддалений репозиторій та безперервна інтеграція

При розробці системи було використано Gitlab у якості віддаленого репозиторію та у якості інструменту для забезпечення безперервної інтеграції.

Безперервна інтеграція — це практика розробки програмного забезпечення, суть якої полягає у тому що члени команди часто інтегрують свою роботу, зазвичай кожна людина інтегрується щонайменше щодня, що призводить до декількох інтеграцій на день. Кожна інтеграція перевіряється автоматизованою збіркою, включаючи модульні тести, щоб виявити помилки інтеграції якнайшвидше [22].

4. ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ СИСТЕМИ АВТОМАТИЗАЦІЇ ЖИТЛОВИХ ПРИМІЩЕНЬ

Реалізація системи автоматизації житлових приміщень передбачає створення ряду обмежених контекстів, створення єдиної мови та моделі предметної області для кожного з обмежених контекстів, реалізація кожного обмеженого контексту у програмному коді та їх інтеграція між собою. Окрім того необхідно реалізувати Web-інтерфейс користувача та інтегрувати його з розробленої системою.

4.1 Обмежені контексти системи автоматизації житлових приміщень

При дослідженні предметної області було виділено наступні обмежені контексти:

— керування акаунтом користувача та конфігурація приміщень;
— моніторинг стану середовища житлового приміщення;
— збір даних пристроїв та формування звітів.

4.1.1 Контекст керування акаунтом користувача та конфігурація приміщень

Глосарій єдиної мови для контексту керування акаунтом користувача та конфігурація приміщень:

— User — користувач системи;
— Room — приміщення, що належить користувачу та придатне до автоматизації;

— Parameter — характеристика стану середовища приміщення. До параметрів входять вологість повітря, температура повітря, освітленість;
— Sensor — вимірювальний пристрій, встановлений у певному приміщенні, вимірює певну характеристику стану середовища приміщення;
— Actuator — регулюючий пристрій, встановлений у певному приміщенні, здатний змінювати стан середовища приміщення по одній або декільком характеристикам;
— Requirement — побажання користувача щодо значення певної характеристики середовища приміщення у певний період часу.

На основі єдиної мови було побудовано програмну модель предметної області (рисунок 4.1).

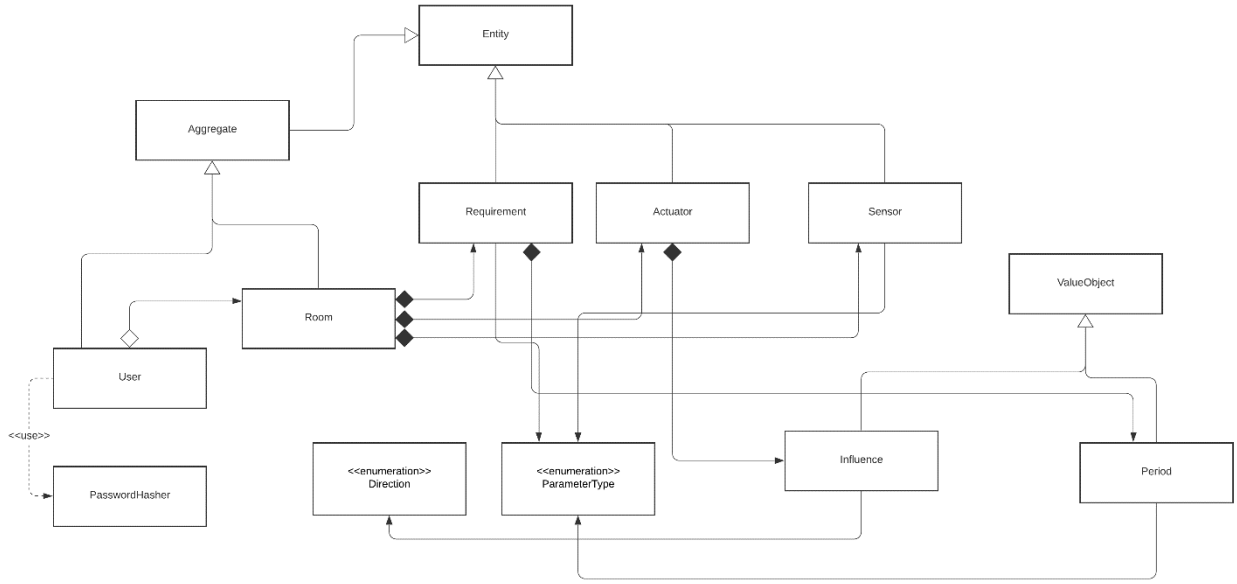

Рисунок 4.1 — Модель предметної області контексту керування акаунтом користувача та конфігурація приміщень

До основних задач розглянутого контексту відноситься забезпечення можливостей користувача створювати та редагувати записи про приміщення для автоматизації, додавати до приміщень вимірювальні та регулюючі прилади тощо.

4.1.2 Контекст моніторингу стану середовища житлового приміщення

Глосарій єдиної мови для контексту моніторингу стану середовища житлового приміщення:

- Environment — середовище, суб'єкт автоматизації;
- EnvironmentState — стан середовища у певний момент, складається з набору значень характеристик середовища;
- Parameter — характеристика стану середовища у певний момент. До параметрів входять вологість повітря, температура повітря, освітленість;
- Temperature — характеристика температури повітря середовища у певний момент. Вимірюється у градусах Цельсія;
- Humidity — характеристика відносної вологості повітря середовища у певний момент. Вимірюється у процентах;
- Illumination — характеристика освітлення середовища у певний момент. Вимірюється у люксах;
- Actuator — регулюючий пристрій, встановлений у середовищі у певний момент. Може бути ввімкненим або вимкненим. Коли ввімкнений — впливає на одну або декілька характеристик середовища.

На основі єдиної мови було побудовано наступну програмну модель предметної області (рисунок 4.2):

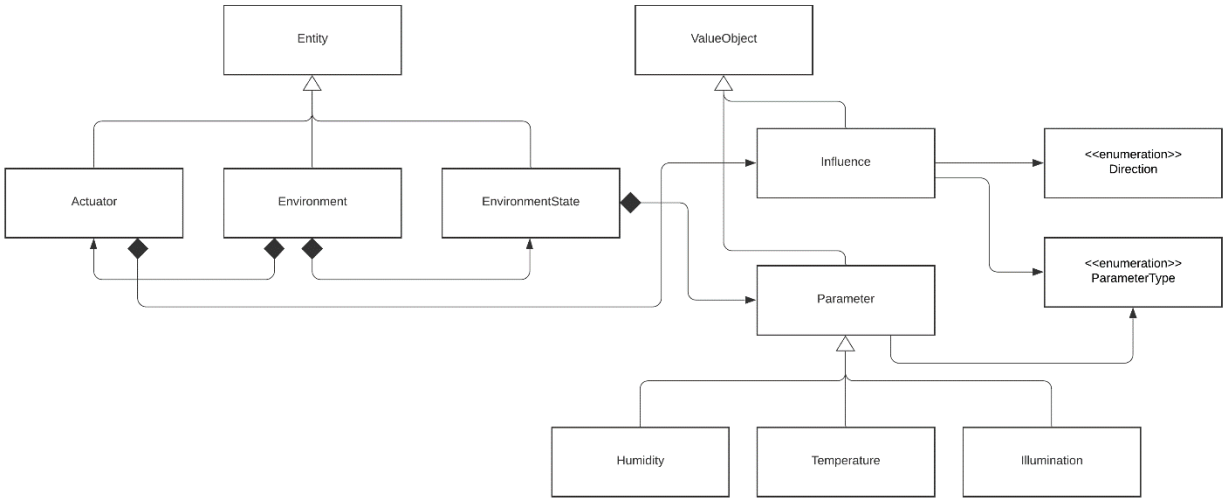

Рисунок 4.2 — Модель предметної області контексту моніторингу стану середовища житлового приміщення

Контекст моніторингу стану середовища житлового приміщення забезпечує аналіз даних з вимірювальних приладів, порівняння поточного стану середовища з бажаним та прийняття рішень про зміну стану регулюючих приладів.

4.1.3 Контекст збору даних пристроїв та формування звітів

Глосарій єдиної мови для контексту збору даних пристроїв та формування звітів:

— Sensor Data — одиничне значення, отримане від певного вимірювального пристрою;

— Actuator State Change — запис, що вказує на факт зміни стану певного регулювального пристрою;

— Room Report — звіт щодо даних, отриманих з вимірювальних та регулювальних пристроїв для певного приміщення у заданий період часу.

На основі єдиної мови було побудовано наступну програмну модель предметної області (рисунок 4.3):

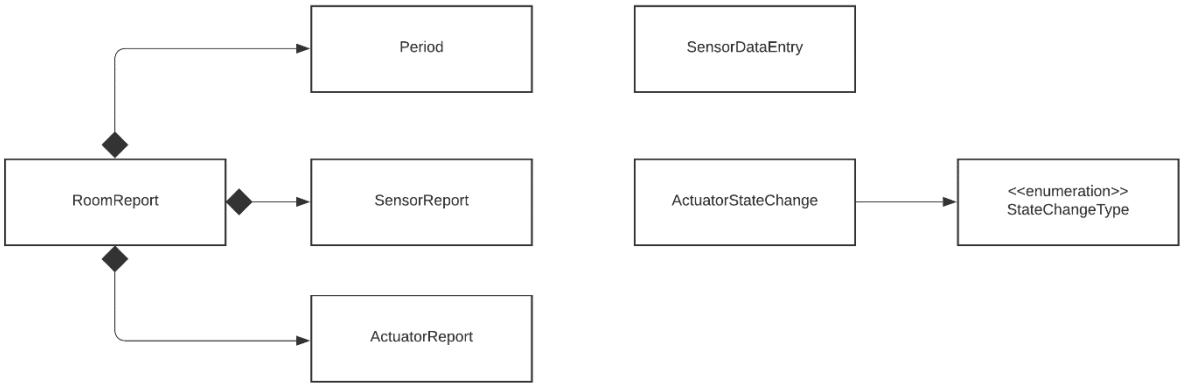

Рисунок 4.3 — Модель предметної області контексту збору даних пристроїв та формування звітів

Контекст збору даних пристроїв та формування звітів не володіє складними правилами предметної області, отже в ньому задля спрощення було упущено розділення об’єктів на сутності та об’єкти значення та групування до агрегатів.

4.2 Програмна реалізація обмежених контекстів системи

4.2.1 Програмна реалізація контексту керування акаунтом користувача та конфігурації приміщень

До реалізації контексту керування акаунтом користувача та конфігурації приміщень входять наступні проекти:

— RoomManagement.Core — містить програмну реалізацію моделі предметної області, логіку додавання приміщень до акаунту користувача та їх конфігурації;

— RoomManagement.Infrastructure — містить інфраструктурні програмні коди для забезпечення функціонування сервісу. Проект відповідальний за збереження даних до бази даних, публікацію повідомлень до брокера повідомлень тощо;
— RoomManagement.Messages — містить набір об’єктів-повідомлень для інтеграції з іншими компонентами системи.

Програмна реалізація контексту являє собою сервіс, що приймає команди та запити від клієнта, обробляє їх та, у разі запиту — повертає результат виконання. Якщо під час обробки команди трапляється значима для предметної області подія, сервіс надсилає про це повідомлення до брокера повідомлень.

4.2.2 Програмна реалізація контексту моніторингу стану середовища житлового приміщення

До реалізації контексту керування акаунтом користувача та конфігурації приміщень входять наступні проекти:

— EnvironmentMonitoring.Core — містить програмну реалізацію моделі предметної області контексту, логіку прийняття рішення щодо керування регулюючими пристроями для конкретного приміщення на базі даних, отриманих з вимірювальних пристроїв;
— EnvironmentMonitoring.Messages — містить набір об’єктів-повідомлень для інтеграції з іншими компонентами системи;
— EnvironmentMonitoring — містить інфраструктурні програмні коди для забезпечення збереження даних до бази даних, комунікації з іншими компонентами системи за допомогою брокера повідомлень.

Програмна реалізація контексту являє собою сервіс, що приймає повідомлення з даними вимірювальних приладів у певний момент, аналізує ці дані та приймає рішення стосовно потреби змінювати стан регулюючих

приладів для конкретного приміщення. В разі потреби змінити стан регулюючого приладу, сервіс надсилає відповідну команду.

4.2.3 Програмна реалізація контексту збору даних пристроїв та формування звітів

Контекст збору даних пристроїв та формування звітів представлений у реалізації програмної системи у вигляді єдиного проекту-сервісу:

— Audit — містить програмну реалізацію моделі предметної області контексту збору даних пристроїв та формування звітів, інфраструктурний код для обробки повідомлень про отримання даних з вимірювальних пристроїв та про зміну стану регулювальних пристроїв, збереження отриманої інформації до бази даних та логіку побудови звітів щодо стану середовища кімнати у певний період часу.

Сервіс обробляє повідомлення про отримання даних з вимірювальних приладів та зміну стану регулювальних приладів, що надходять з брокеру повідомлень. Сервіс також обробляє запити від Web API на побудову звітів по отриманим даним.

4.3 Структура програмної реалізації системи

Програмна реалізація компонентів системи передбачає створення набору сервісів та бібліотек програмного коду. Далі представлена структура проектів програмної реалізації системи (рисунок 4.4):

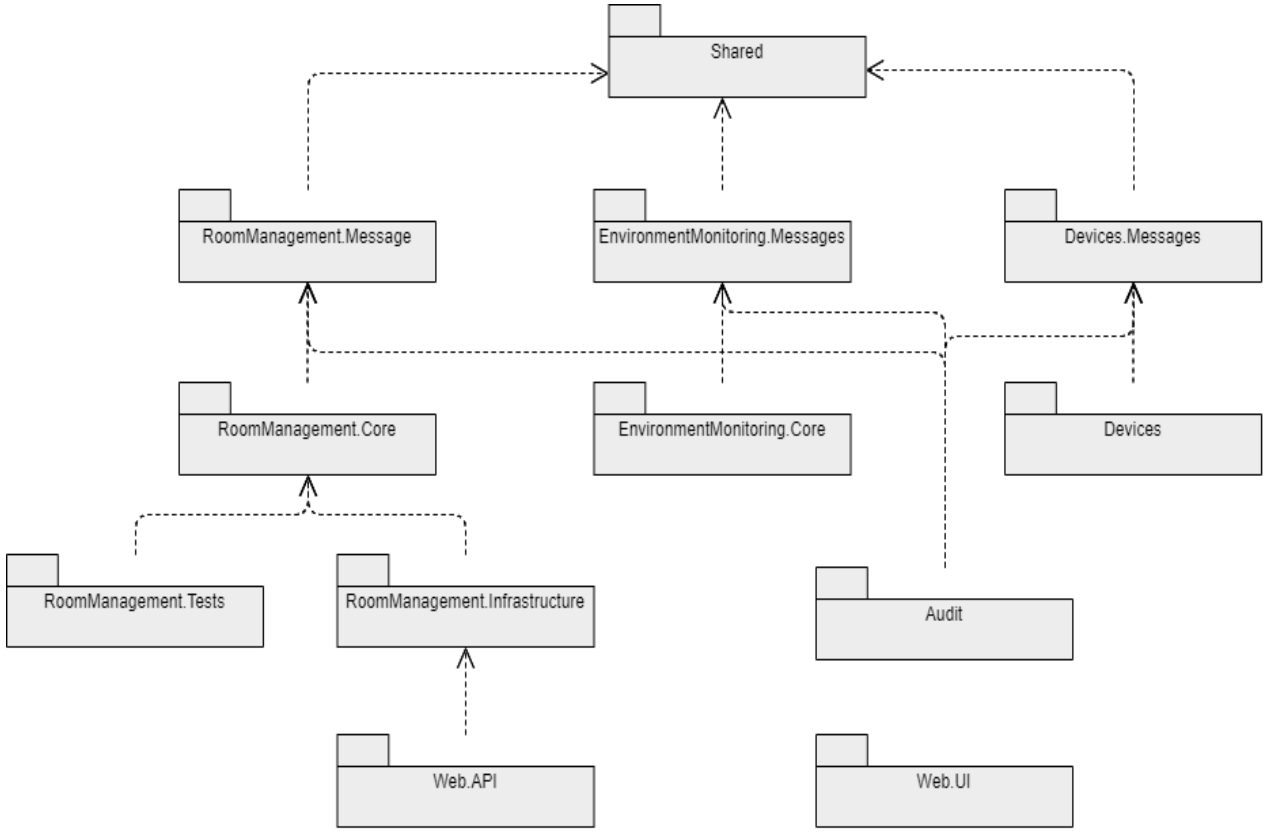

Рисунок 4.4 — Діаграма залежностей проектів програмної реалізації системи

Більшість проектів програмної реалізації іменовано згідно з обмеженим контекстом, до котрого ці проекти відносяться. Слід зауважити, що код певних проектів виконується у окремих процесах та у вигляді окремих сервісів, отже між проектами існують додаткові залежності, викликані інтеграцією декількох сервісів за допомогою відповідних протоколів.

Обмежені контексти реалізовано у вигляді мікросервісів. Кожен з мікросервісів використовує власну базу даних. Зміни розповсюджуються між мікросервісами шляхом публікації подій за допомогою брокера повідомлень RabbitMQ та обробки цих подій у мікросервісах, що на них підписані. У якості Web-інтерфейсу системи виступає Web API на базі технології ASP.NET Core. Дане Web API є шлюзом для серверної частини системи та відповідає за загальні потреби Web-орієнтованих програмних систем, такі як впровадження

схеми автентифікації та авторизації, обробку HTTP-запитів та перенаправлення цих запитів до обмежених контекстів у вигляді контрактів. Схематичне зображення високорівневої структури системи приведено далі (рисунок 4.5):

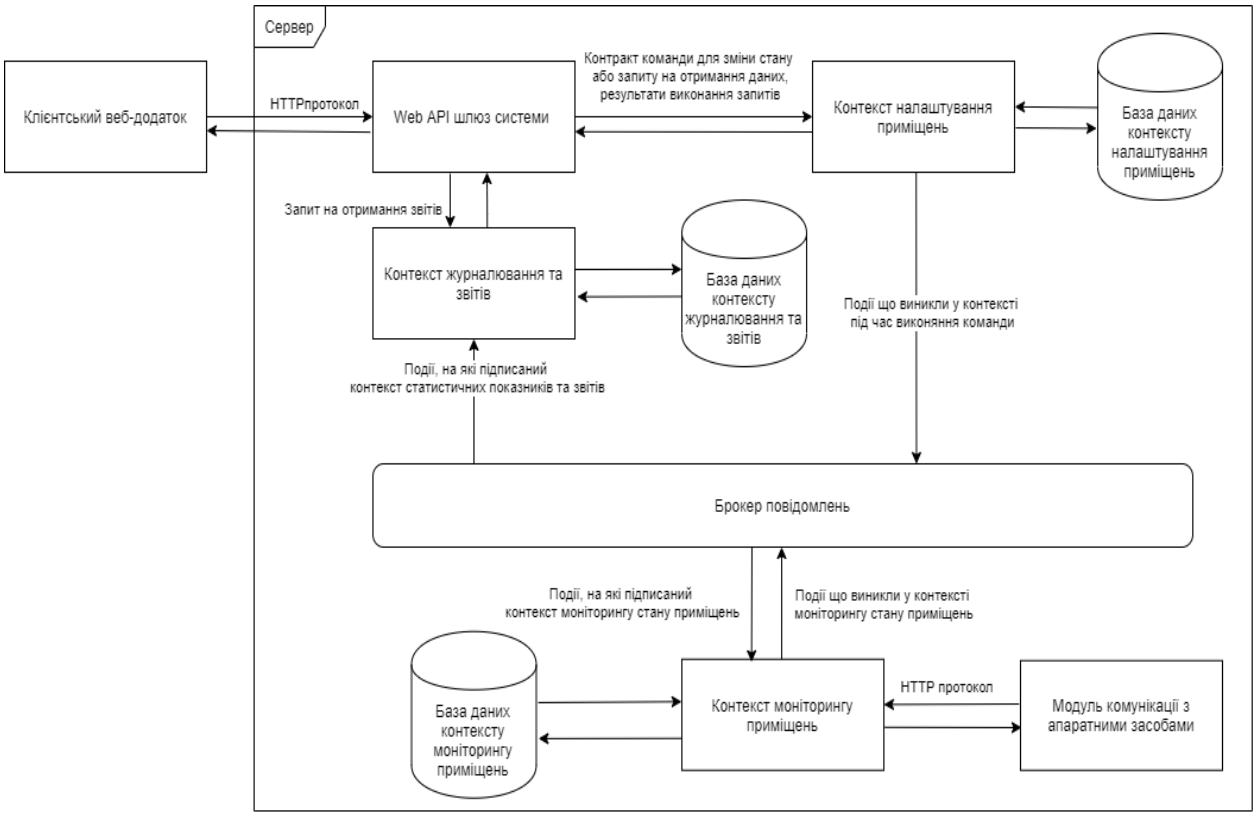

Рисунок 4.5 — Високорівнева структура системи

У поточній реалізації системи присутній інтерфейс користувача у вигляді Web-застосунку, розробленого но мові JavaScript за допомогою бібліотеки React. Проте наявність Web API шлюзу для серверної частини системи передбачає можливість створення альтернативних рішень для забезпечення інтерфейсу користувача. До таких альтернативних рішень може відноситись будь який програмний застосунок, що має можливість комунікації за допомогою протоколу HTTP.

4.4 Реалізація серверної частини системи

Серверна частина системи складається з трьох обмежених контекстів представлених у вигляді мікросервісів та Web API шлюзу для забезпечення єдиної точки доступу до серверу з боку клієнтської частини.

Для реалізації контексту керування акаунтом користувача та конфігурації приміщень та контексту моніторингу стану середовища житлового приміщення використовувалась наступна схема взаємодії шарів застосунку (рисунок 4.6):

Рисунок 4.6 — Діаграма залежності шарів програмної реалізації обмеженого контексту

Дана організація шарів застосунку передбачає наступне функціональне призначення кожного з шарів:

— контракти задають інтерфейс взаємодії обмеженого контексту з іншими компонентами системи;
— модель предметної області містить бізнес-правила певного обмеженого контексту, основну логіку його роботи;
— інфраструктура забезпечує потреби обмеженого контексту, що не стосуються логіки предметної області, таки як: робота з базою даних, інтеграція з брокером повідомлень, забезпечення реалізації бізнес-транзакцій тощо;
— сервіси рівня застосунку реалізують певні контракти обмеженого контексту, що часто співпадають з прецедентами системи. До обов’язків шару сервісів рівня застосунку є створення об’єктів предметної області та виклик на цих об’єктах методів з метою реалізації функціональності, передбаченої контрактом;
— шар відображення слугує інтерфейсом обмеженого контексту, у випадку даної системи, шар відображення реалізовано як Web API.

Зазначений спосіб організації шарів застосунку передбачає використання принципу інверсії залежностей. Якщо у шарі моделі предметної області потребується доступ до будь-яких інфраструктурних компонентів, визначається інтерфейс, котрий в подальшому має бути реалізований шаром інфраструктури. Такий підхід є притаманним для реалізації систем, що розроблялися згідно принципам предметно-орієнтованого проектування.

Контекст збору даних пристроїв та формування звітів в свою чергу реалізований значно простіше. Проект не має чітко визначених шарів та являє собою застосунок для збереження та зчитування даних з бази даних. Таке рішення аргументовано простотою даного контексту у створеному прототипі системи автоматизації споруд.

Далі у якості прикладу представлено структуру програмної реалізації контексту керування акаунтом користувача та конфігурації приміщень

(рисунок 4.7). Слід зазначити, що проект, котрий реалізує шар контрактів написаний з використанням мови F# з метою використання типів-записів — незмінних структур даних з підтримкою вбудованої перевірки на структурну рівність.

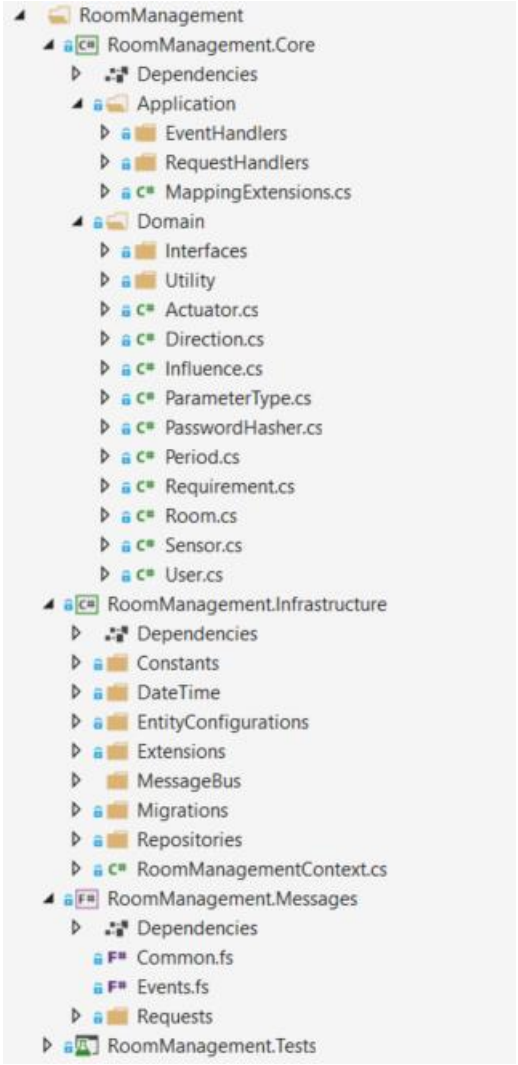

Рисунок 4.7 – Структура програмної реалізації контексту керування акаунтом користувача та конфігурації приміщень

Зазначимо, що в даній реалізації було об’єднано шари моделі предметної області та сервісів рівня додатку до однієї збірки — RoomManagement.Core.

Контракти та інфраструктура при цьому винесені до окремих збірок — RoomManagement.Infrastructure та RoomManagement.Messages.

В якості технології для доступу до даних було використано модуль об’єктно-реляційного відображення Entity Framework Core, що підтримує створення моделі бази даних з існуючої об’єктної моделі на мові C#. Отже, модель бази даних відповідає моделям обмежених контекстів, приведених раніше. Далі в якості прикладу буде розглянуто модель відношення сутностей контексту керування акаунтом користувача та конфігурації приміщень (рисунок 4.8).

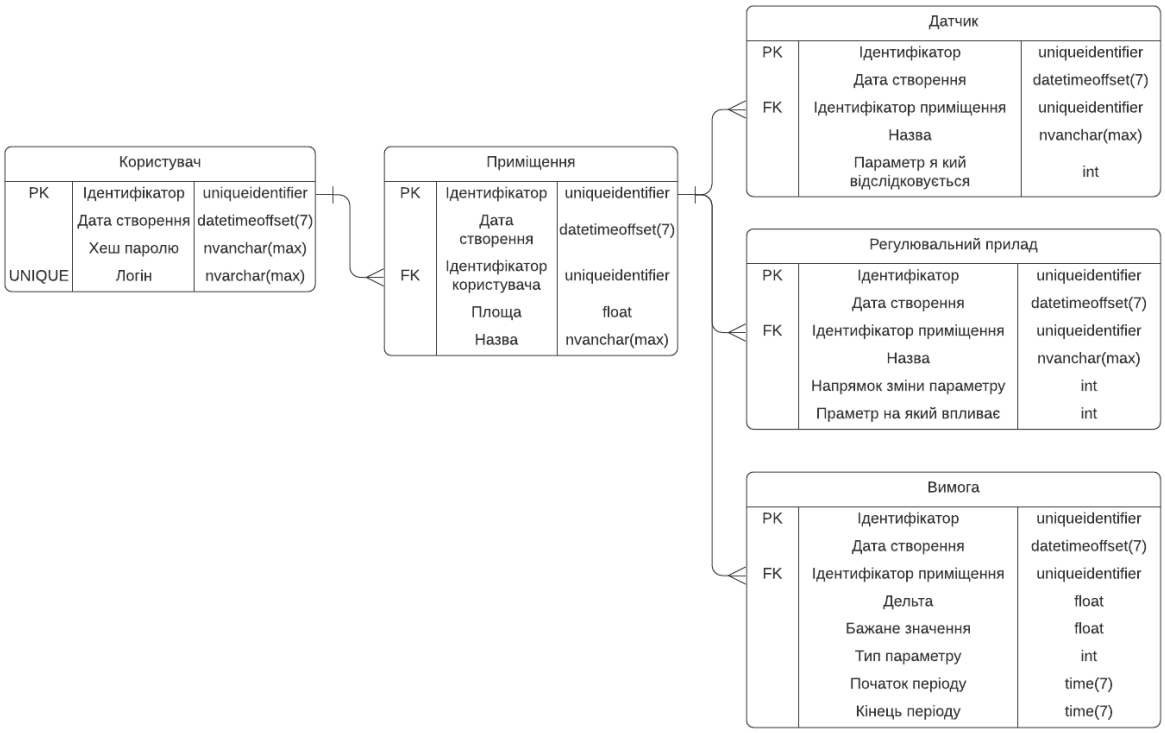

Рисунок 4.8 — Діаграма відношення сутностей бази даних контексту керування акаунтом користувача та конфігурації приміщень

Приведена модель дещо відрізняється від об’єктної моделі відповідного обмеженого контексту за рахунок відображення об’єктів-значень до моделі бази даних у вигляді вбудованих до сутності-власника значень. Такий підхід

до відображення об’єктів-значень краще відповідає їхньому призначенню та властивостям.

Розглянемо загальний алгоритм обробки контракту обмеженим контекстом на прикладі діаграми послідовностей для процесу додавання користувачем вимоги до приміщення (рисунок 4.9):

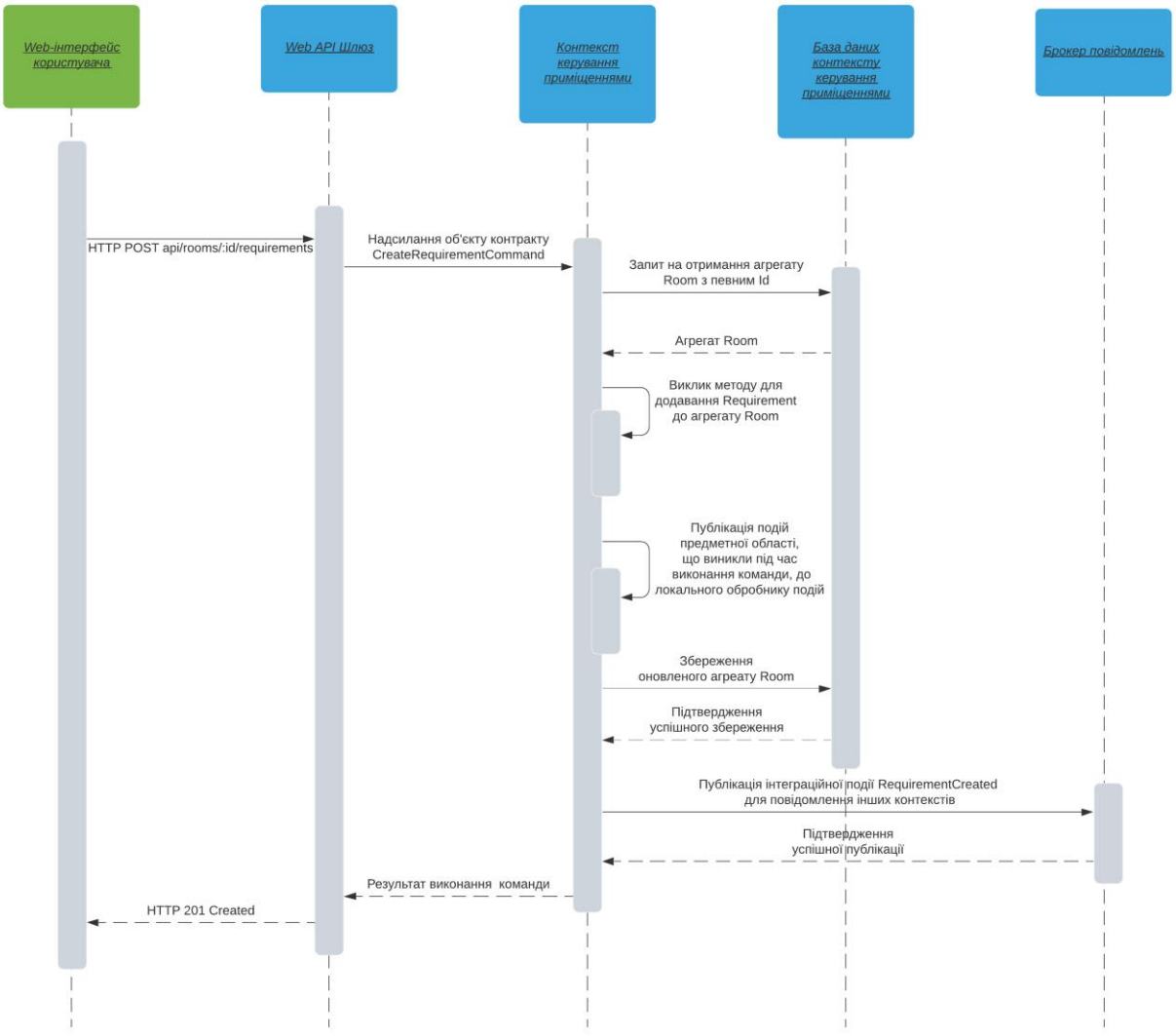

Рисунок 4.9 — Діаграма послідовностей процесу додавання користувачем вимоги до певного приміщення

Слід зазначити, що в реалізованій системі використано підхід до обробки подій предметної області у тій самій транзакції бази даних, в якій вони

були ініційовані. На діаграмі це зазначено послідовним викликом публікації подій предметної області до локального обробнику перед виконанням збереження агрегату до бази даних. Можливий також альтернативний варіант обробки кожної події у власній транзакції, який у поточній системі не було реалізовано.

4.5 Реалізація клієнтської частини системи

Клієнтська частина реалізована як Web-додаток. При розробці використовувалась мова програмування JavaScript та бібліотека React. Для створення візуальної частини інтерфейсу було використано React Material UI — бібліотеку, що містить набір готових компонентів React для пришвидшення побудови інтерфейсів. Структура клієнтського додатку зображена далі (рисунок 4.10):

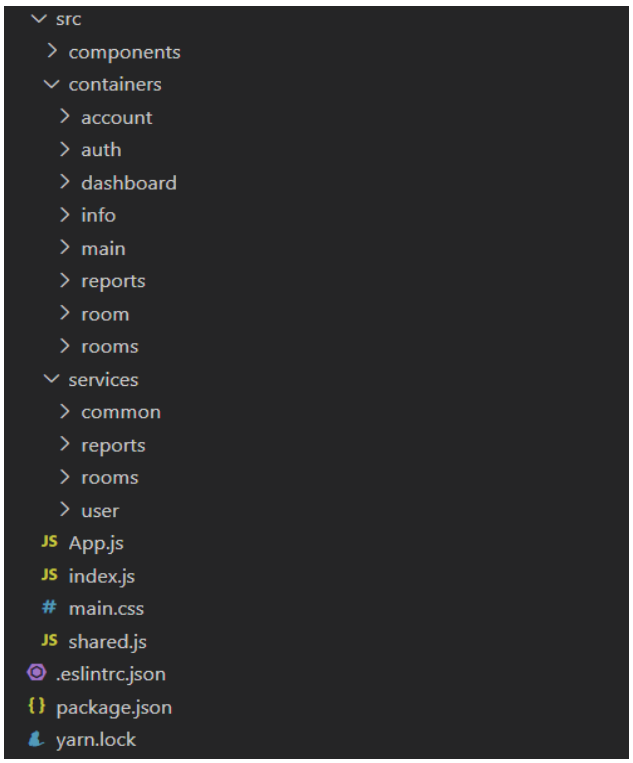

Рисунок 4.10 — Структура програмної реалізації клієнтського додатку

Як видно із приведеної структури, проект розділено на компоненти, контейнери та сервіси.

Компонент — це елемент користувацького інтерфейсу, здатний до повторного використання. В даному проекті до компонентів відносяться різноманітні елементи форм для введення даних про приміщення, компоненти для зображення графіків та таблиць тощо.

Контейнер — це комбінація певних компонентів, що утворює цілісну сторінку користувацького інтерфейсу. В проекті реалізовано сторінки для автентифікації, реєстрації, керування аккаунтом, створення та налаштування приміщень тощо.

Сервіси в контексті клієнтського додатку — набір об’єктів та функцій для налагодження комунікації із серверною частиною системи. Поточна реалізація передбачує надсилання HTTP запитів до Web API. Для надсилання HTTP запитів використано бібліотеку axios.

Для керування залежностями клієнтського додатку було використано менеджер залежностей yarn, що, в свою чергу, використовує репозиторій пакетів npm.

.

5. РОБОТА КОРИСТУВАЧА З ПРОГРАМНОЮ СИСТЕМОЮ

Для кінцевого користувача система представляє собою веб-застосунок без необхідності встановлення жодного програмного забезпечення. Інтерфейс веб-застосунку буде розглянуто далі у цьому розділі. Серверна частина системи представлена множиною сервісів, що можуть бути запущені вручну або за допомогою системи керування контейнерами Docker.

5.1 Системні вимоги

Система автоматизації житлових приміщень являє собою серверне програмне забезпечення. Залежності та налаштування середовища виконання програмної реалізації системи інкапсульовані у Docker образах для кожного з відповідних модулів системи та у Docker Compose конфігурації. Для функціонування контейнерів системи на певному хості потребується наявність Docker Engine. Рекомендована версія Docker Engine — 19.03.0 або вище. Для розгортання системи на певному хості має бути наявним інструмент Docker Compose версії 1.21.0 або вище.

5.2 Опис роботи користувача з системою

Web-інтерфейс надає користувачам можливість керування системою автоматизації житлових приміщень. Web-інтерфейс являє собою Web-сайт з можливостями авторизації в системі, створення нового акаунту, редагування даних акаунту, створення записів про приміщення, придатні до автоматизації,

встановлення бажаних значень для різноманітних характеристик середовища житлового приміщення, конфігурації приміщень набором вимірювальних та регулювальних пристроїв тощо.

Також окрема сторінка Web-сайту відповідає за відображення звітів щодо стану середовища певного приміщення у заданий період часу. До звіту входять графіки значень вимірюваних характеристик певного приміщення, інформація про періоди активності регулювальних приладів, статистичні дані роботи системи для певного приміщення.

Схема роботи користувача з системою представлена на наступній діаграмі прецедентів (рисунок 5.1):

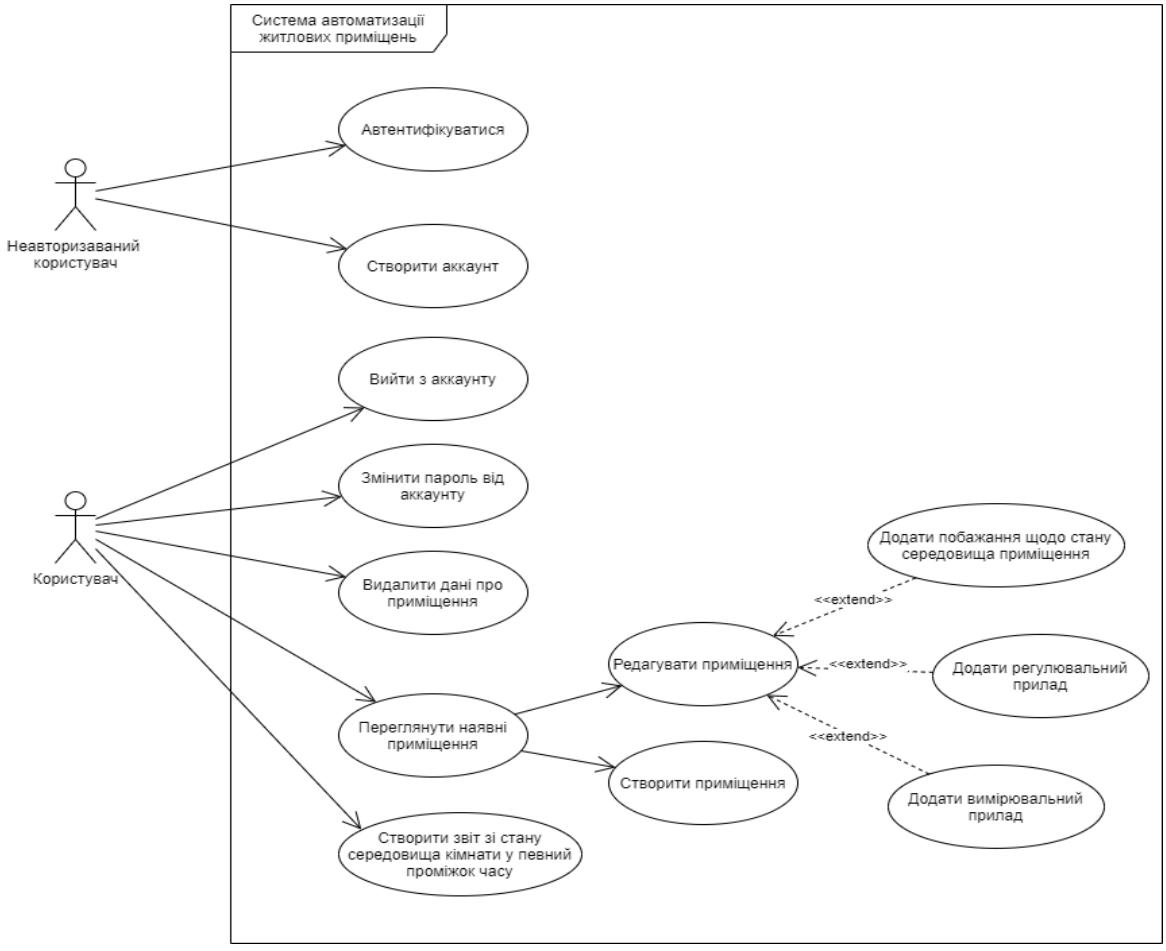

Рисунок 5.1 — Діаграма прецедентів системи автоматизації житлових приміщень

Перед початком користування системою користувач має увійти до свого акаунту. Якщо користувач ще не має свого акаунту, для початку роботи з системою йому пропонується зареєструвати новий акаунт. Сторінку реєстрації нового акаунту зображено на скріншоті (рисунок 5.2):

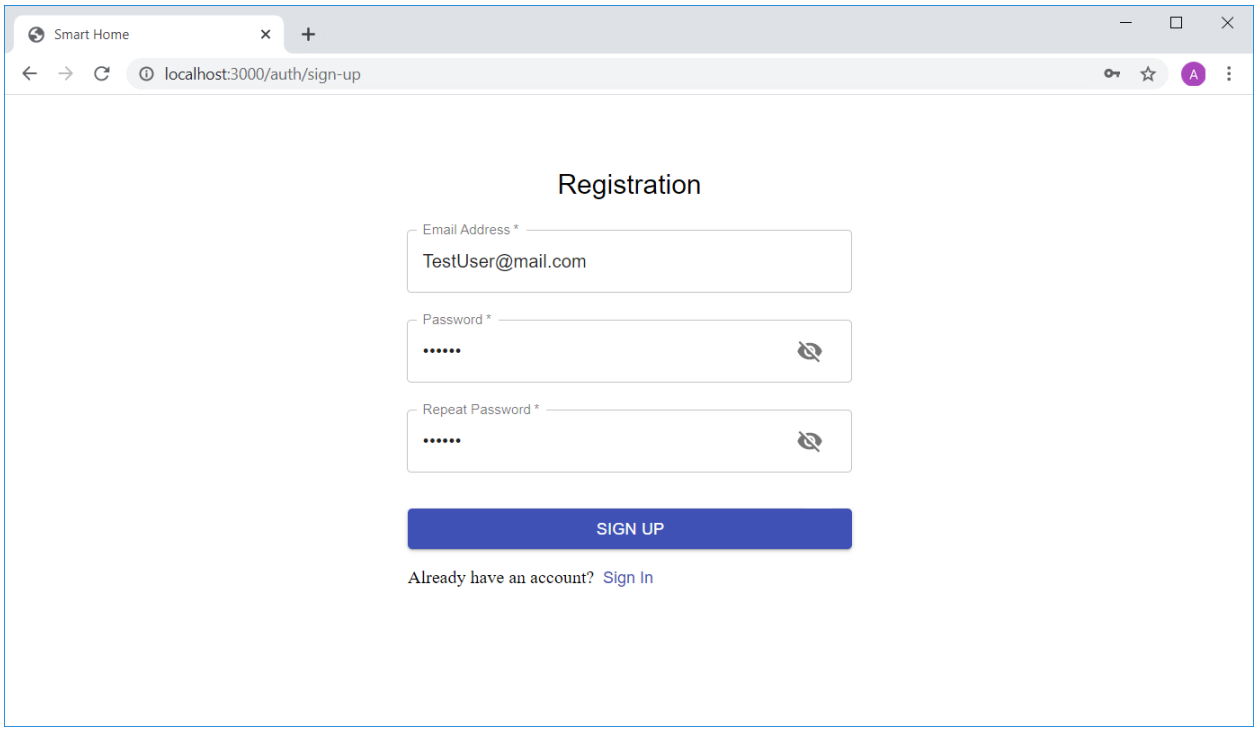

Рисунок 5.2 — Сторінка реєстрації нового акаунту у системі

Після створення акаунту, користувач має змогу автентифікуватись у системі, ввівши адресу електронної пошти, вказану при реєстрації та пароль. Процес авторизації користувача зображено на скріншоті (рисунок 5.3):

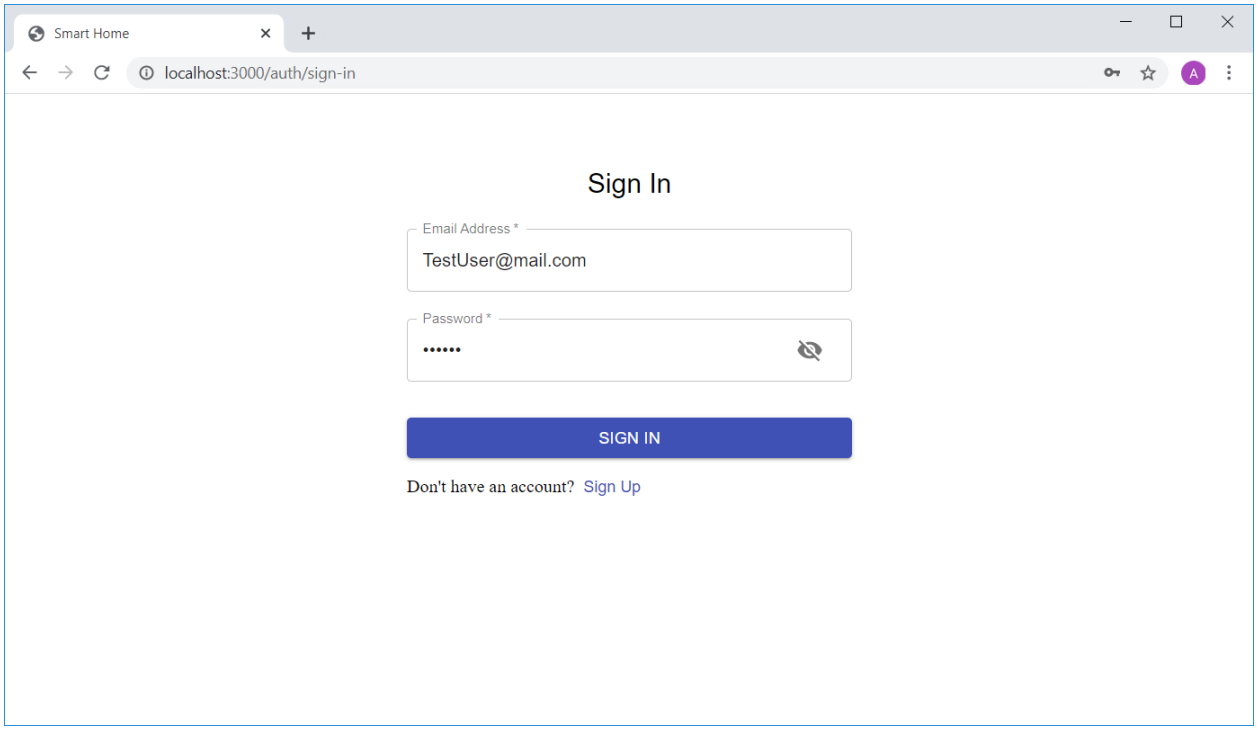

Рисунок 5.3 — Сторінка автентифікації користувача у системі

Після входу до свого акаунту, користувач отримує доступ до панелі керування системою автоматизації житлових приміщень. Панель керування системою налічує наступні сторінки:

— сторінка для керування наявними приміщеннями;
— сторінка для створення та перегляду звітів щодо певного приміщення;
— сторінка для перегляду інформації про акаунт користувача та його налаштування;
— сторінка для перегляду загальної інформації про систему.

Сторінку перегляду та редагування списку приміщень відображає список наявних приміщень користувача, та короткий опис кожного з приміщень. При бажанні користувач має змогу перейти до детального перегляду інформації про певне приміщення та до налаштувань бажаного стану приміщення та наявних вимірювальних і регулюючих приладів або видалити дані про певне приміщення з системи. На сторінці наявна кнопка для

створення нового приміщення. Сторінку керування приміщеннями користувача зображено на скріншоті (рисунок 5.4):

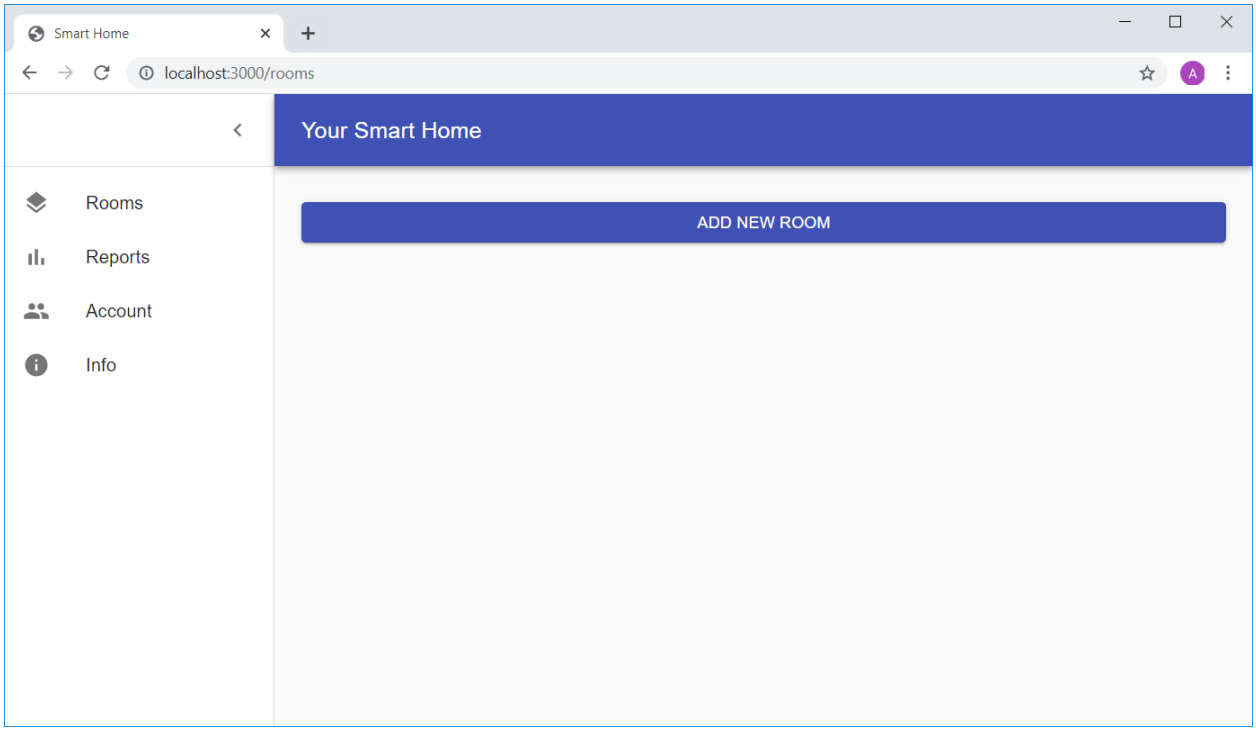

Рисунок 5.4 — Сторінка керування приміщеннями користувача

При натисканні на кнопку створення нового приміщення, користувачеві пропонується заповнити форму створення нового приміщення (рисунок 5.5). На формі наявні поля для введення назви створюваного приміщення та площі приміщення:

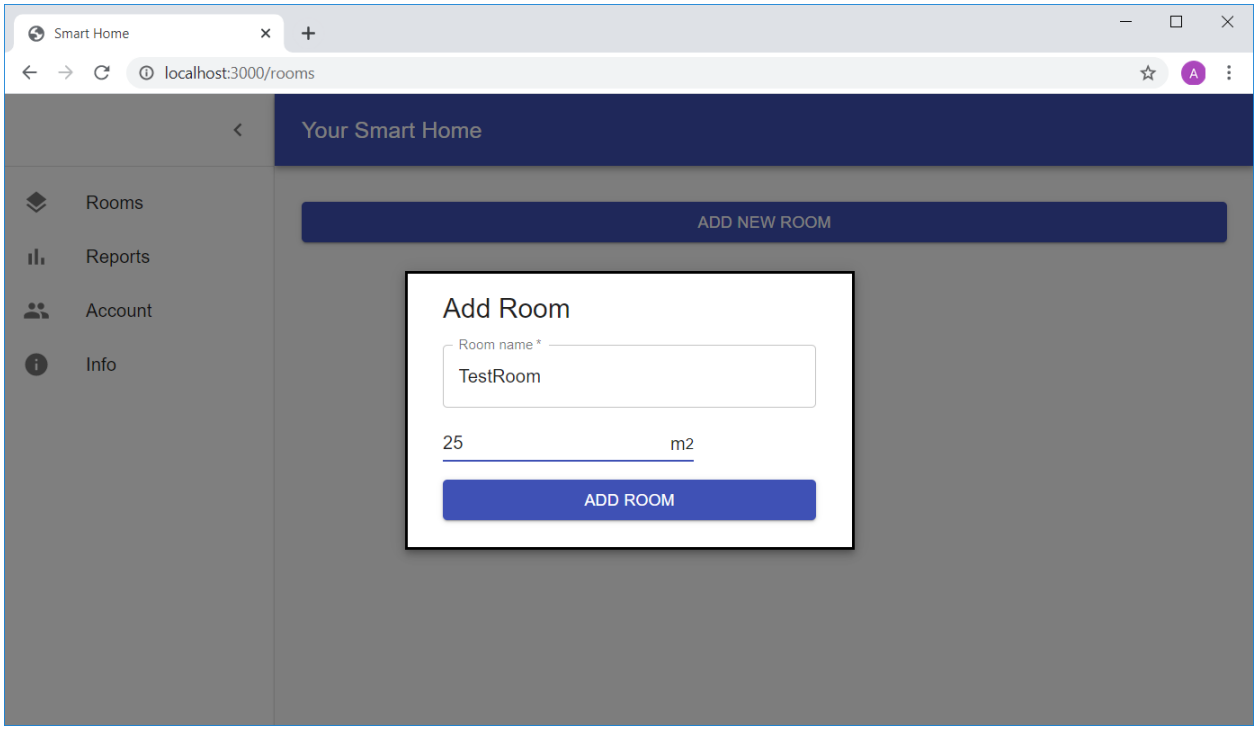

Рисунок 5.5 — Форма створення нового приміщення

Після створення приміщення, воно додається до списку наявних приміщень користувача (рисунок 5.6):

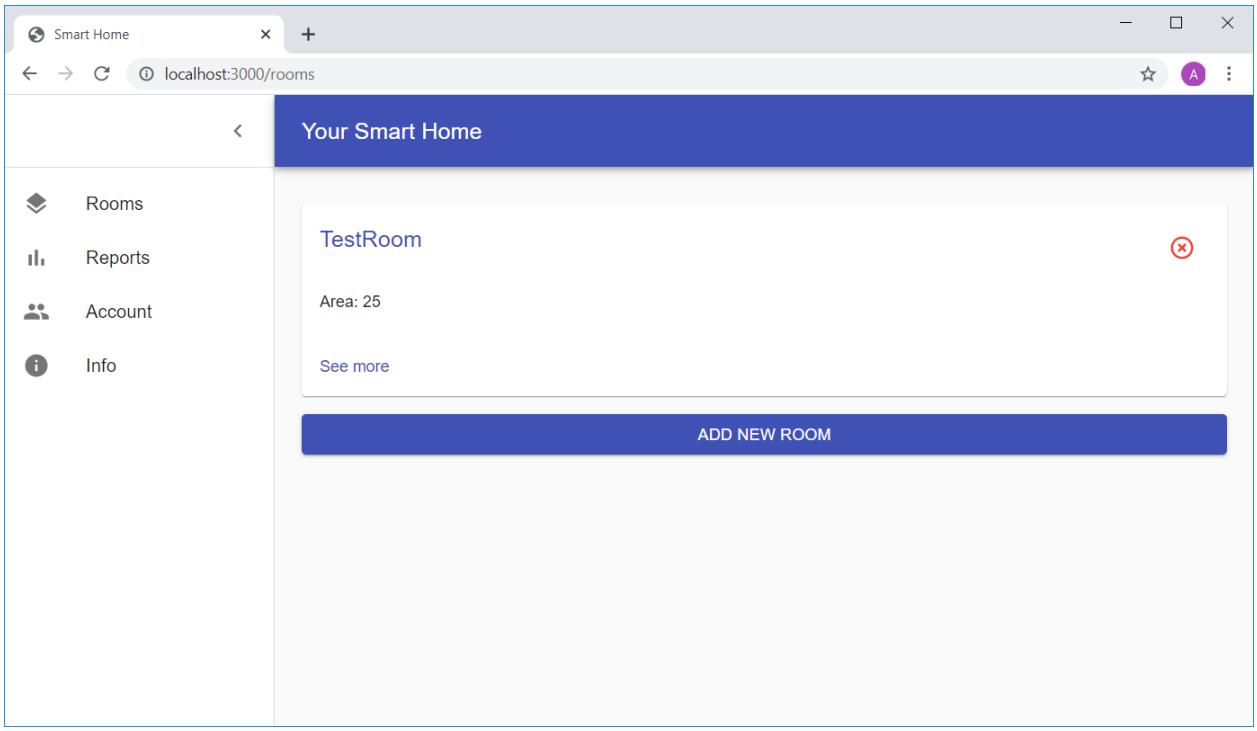

Рисунок 5.6 — Список наявних приміщень користувача з щойно створеним приміщенням

Після створення приміщення користувач має змогу перейти до детального налаштування даного приміщення. Сторінка налаштування певного приміщення надає змогу конфігурувати бажаний стан середовища для певного приміщення, додавати вимірювальні та регулювальні прилади тощо (рисунок 5.7).

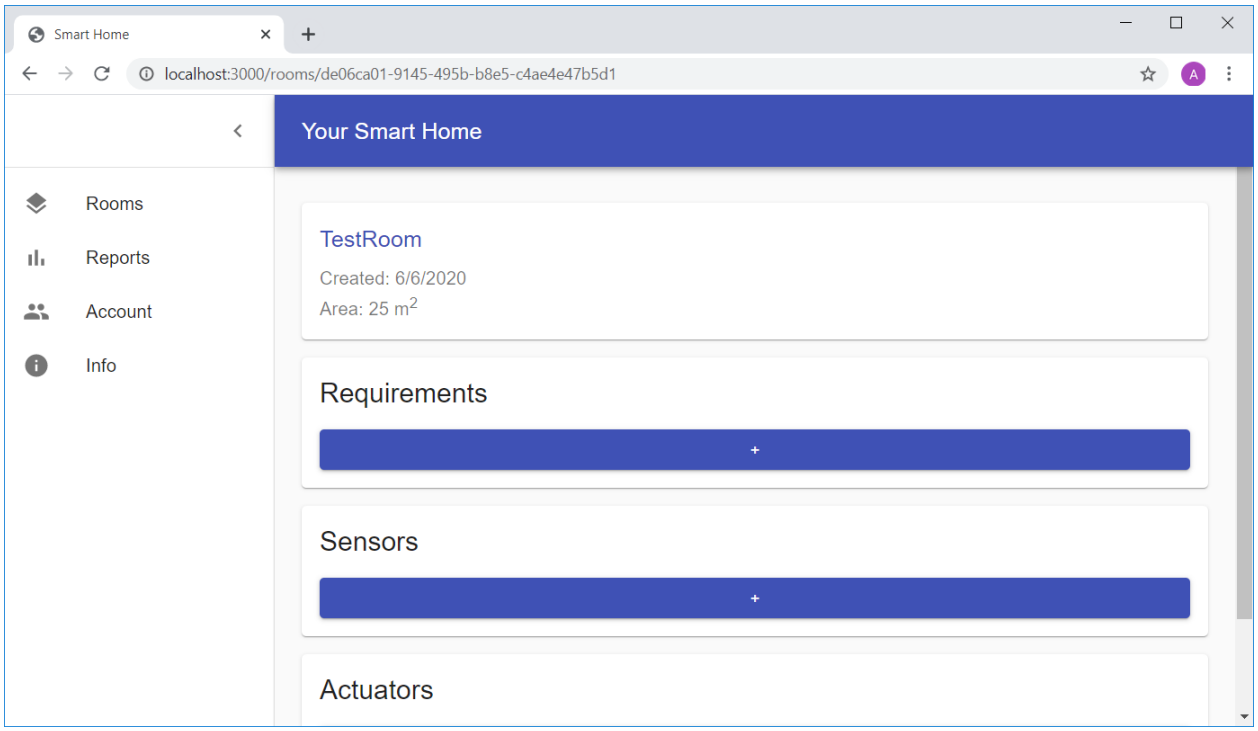

Рисунок 5.7 — Сторінка перегляду інформації про певне приміщення

Система надає можливість користувачеві додавати побажання щодо стану приміщення у певний проміжок часу протягом доби. Для цього користувач має заповнити форму, що складається з наступних полів вводу:

— початок проміжку часу протягом доби, для котрого користувач бажає додати побажання;

— кінець проміжку часу протягом доби, для котрого користувач бажає додати побажання;

— параметр щодо значення котрого відноситься побажання користувача;

— бажане значення заданого параметру у відповідних одиницях вимірювання;

— похибка у значенні обраного параметру, котру користувач вважає не істотною та погоджується з тим, що регулювальні пристрої не будуть реагувати на відходження вимірюваного параметру від бажаного значення на величину, меншу за дану похибку.

Форма додавання побажання щодо стану приміщення зображена на скріншоті (рисунок 5.8):

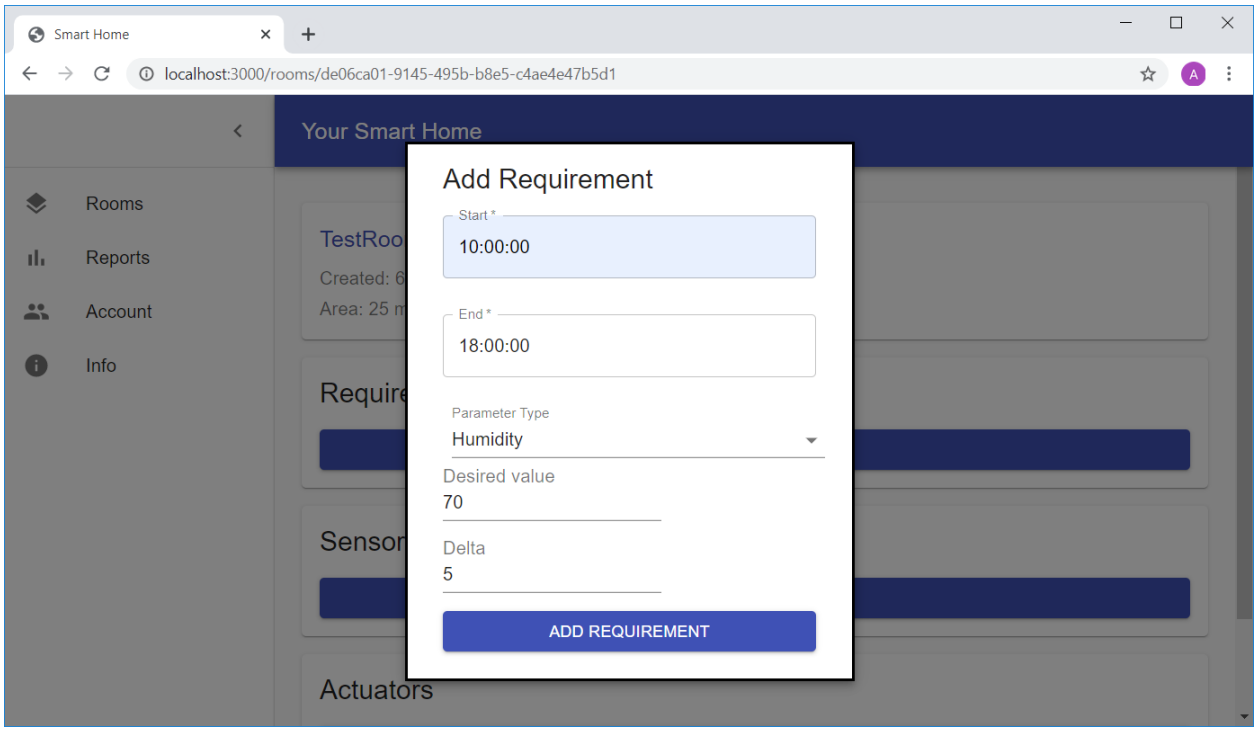

Рисунок 5.8 — Форма додавання побажання щодо стану приміщення

Окрім побажань щодо стану приміщення у певний проміжок часу, на сторінці перегляду певної кімнати користувач має можливість додати

вимірювальні та регулювальні прилади. Форма додавання вимірювального приладу складається з наступних полів (рисунок 5.9):

— назва вимірювального приладу;

— параметр, значення котрого вимірює даний прилад.

До форми додавання регулювального приладу входять наступні поля (рисунок 5.10):

— назва регулювального приладу;

— параметр, на котрий даний прилад може впливати;

— напрямок впливу даного приладу на заданий параметр.

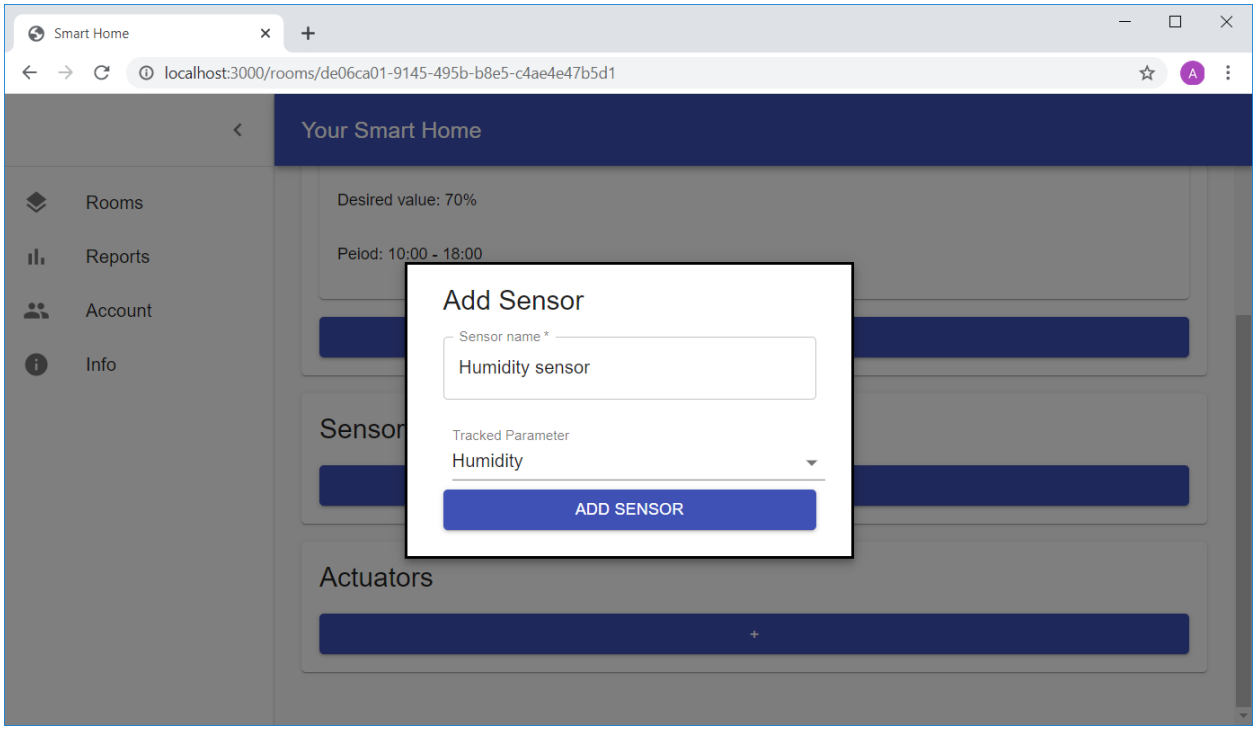

Рисунок 5.9 — Форма додавання вимірювального приладу

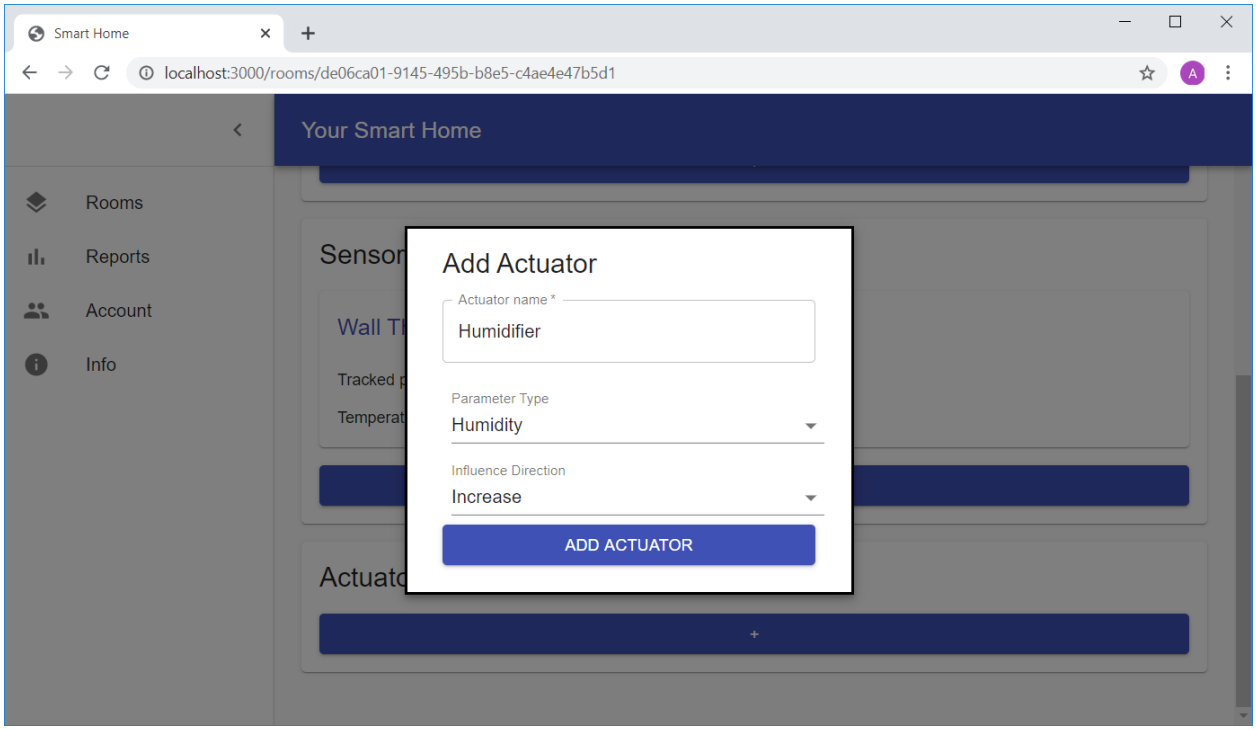

Рисунок 5.10 — Форма додавання регулювального приладу

Після додавання кімнати та налаштування приладів та побажань користувача, система починає процес регулювання стану приміщення відповідно до побажань користувача за допомогою вказаних приладів.

Дана робота не передбачує інтеграції системи з реальними приладами, отже в якості даних вимірювальних приладів використовуються журнальні записи щодо значень певних параметрів середовища у приміщенні. Регулювальні ж прилади в свою чергу існують лише у вигляді програмних об’єктів.

При подальшому користуванні програмною системою користувач має змогу переглянути динаміку зміни вимірюваних параметрів у приміщенні, періоди активності регулювальних приладів тощо. Для цього користувачеві пропонується створити звіт щодо певного приміщення на сторінці звітів (рисунок 5.11):

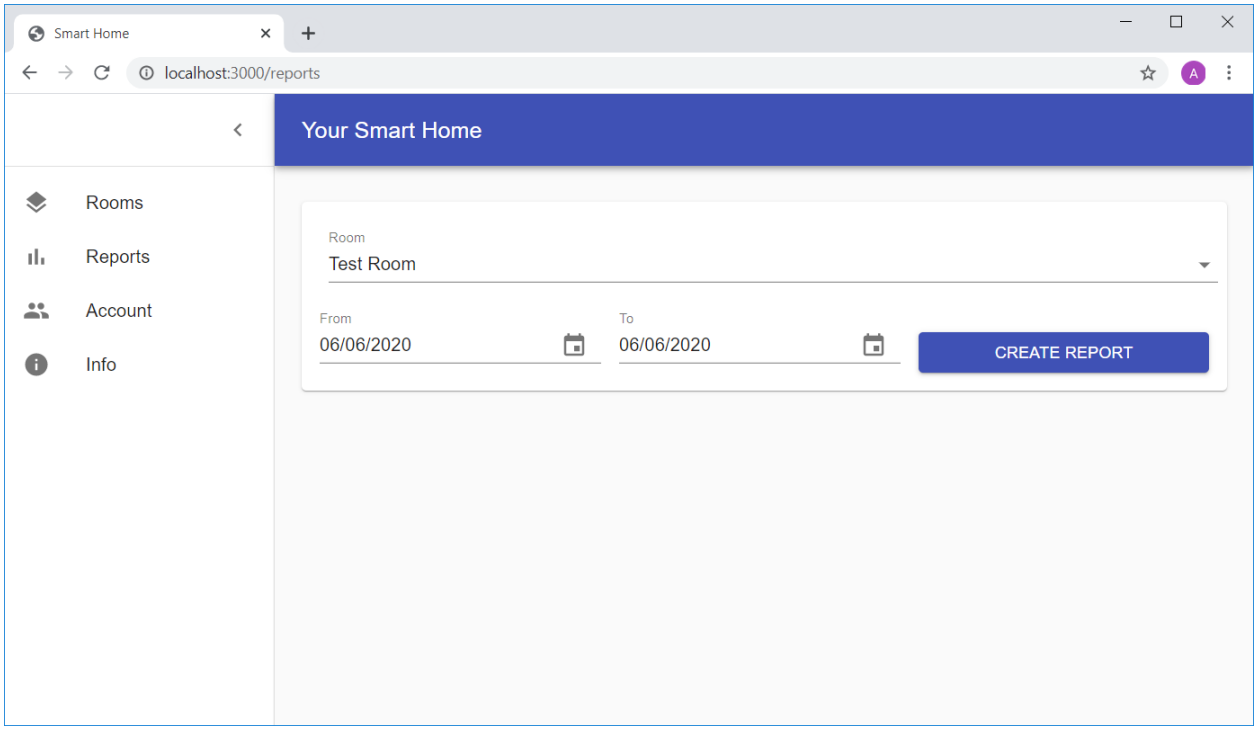

Рисунок 5.11 — Сторінка створення звітів щодо певного приміщення

На сторінці користувач може обрати одне з наявних приміщень та період за який необхідно отримати звіт. Після запиту від користувача, система генерує звіт, що зображує стан приміщення протягом обраного періоду часу за наявності даних з вимірювальних приладів. Також до звіту входять записи про надсилання команд до регулюючих приладів.

На скріншоті (рисунок 5.12) зображено звіт щодо стану заданої кімнати. На скріншоті (рисунок 5.13) зображено звіт з команд, надісланих до регулювальних приладів. Адже створена система не передбачає інтеграції з реальними вимірювальними та регулювальними приладами, дані, отримані у звіті базуються на журнальних записах про показники вимірювальних приладів. Дані про команди, надіслані до регулювальних приладів, базуються на отриманих журнальних значеннях та побажаннях користувача стосовно стану житлового приміщення.

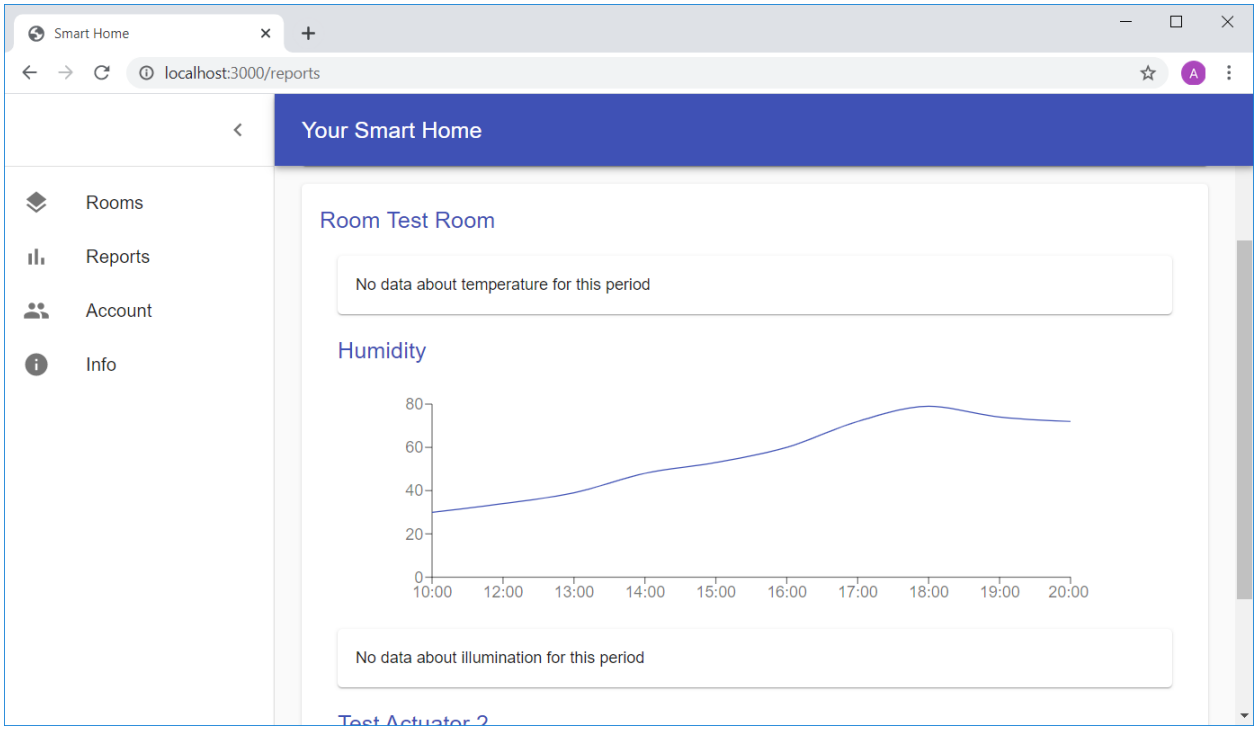

Рисунок 5.12 — Звіт щодо стану приміщення за заданий проміжок часу

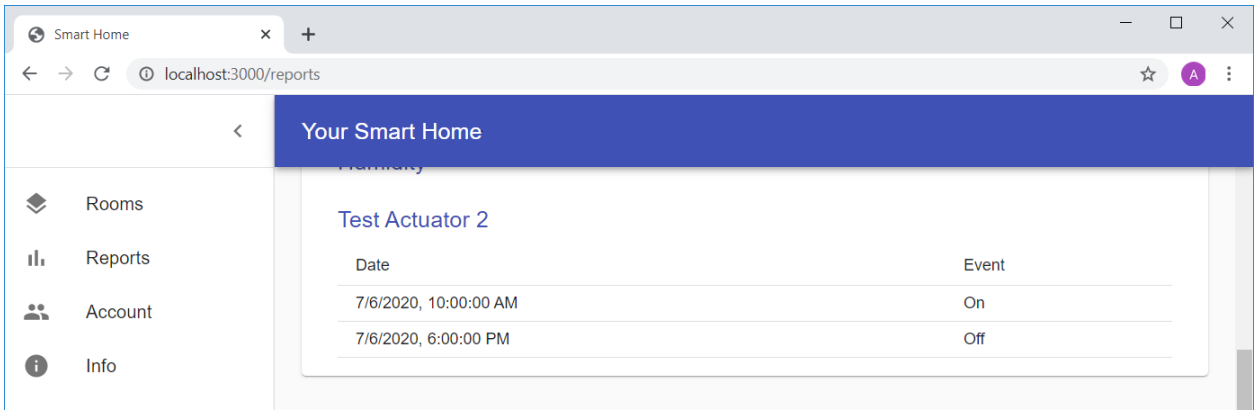

Рисунок 5.13 — Звіт щодо команд регулюючим приладам у певний проміжок часу

Сторінка керування акаунтом надає можливість користувачу передивитись інформацію стосовно свого акаунту, змінити пароль від акаунту та вийти з системи. Сторінку перегляду акаунту користувача зображено на скріншоті (рисунок 5.14):

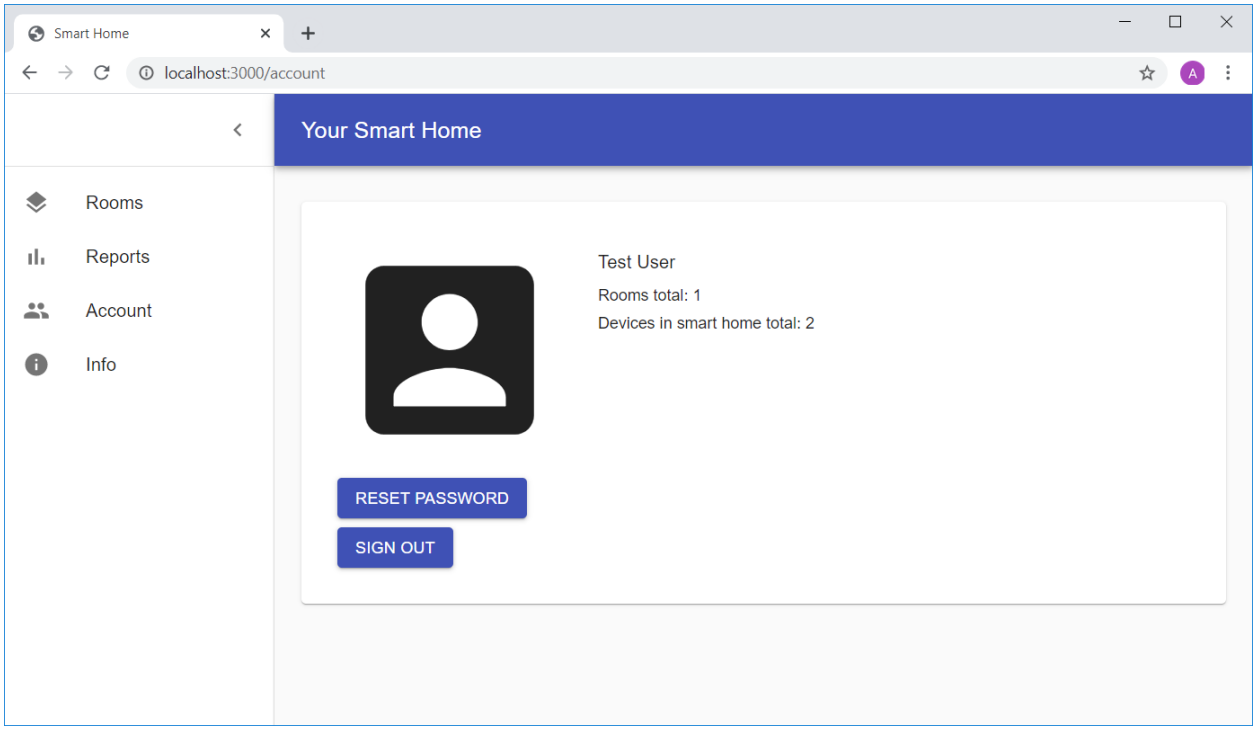

Рисунок 5.14 — Сторінка перегляду акаунту користувача

За рахунок того що серверна частина системи реалізує публічний відкритий Web-інтерфейс у вигляді Web API, стає можливим створення сторонніх клієнтських додатків для забезпечення варіативного представлення клієнтської частини системи. Наприклад, інтерфейс користувача для мобільних систем на базі Android або IOS (у поточній роботі не представлено).

ВИСНОВКИ

Під час виконання роботи було сформульовано задачу розробки системи автоматизації житлових приміщень з використанням підходів предметно-орієнтованого проектування. Відповідно до задачі було проаналізовано предметну область автоматизації житлових приміщень, розглянуто основні принципи та підходи до проектування та реалізації подібних систем. З метою конкретизації задачі було проаналізовано три аналогічні до поточної системи програмні рішення, встановлено їх переваги та недоліки. Було проведено огляд предметно-орієнтованого проектування, розглянуто основні стратегічні та тактичні підходи до проектування за допомогою даного підходу, досліджено основні джерела з даної теми. На основі отриманих знань було спроектовано поточну систему автоматизації житлових приміщень – виявлено обмежені контексти наведеної предметної області, складено глосарії єдиної мови для кожного с обмежених контекстів.

Для створення ефективної програмної реалізації було досліджено ряд технологій та засобів розробки, обрано відповідний стек технологій для реалізації системи. Серед обраних технологій, мов та засобів розробки можна виділити мову програмування C#, платформу .NET Core, реляційну базу даних SQL Server, середу розробки Visual Studio, фреймворк для побудови Web-додатків ASP.NET Core, засіб інтеграції програмних підсистем — брокер повідомлень RabbitMQ. Під час реалізації програмної системи було отримано практичні навички з використання вказаних технологій, мов програмування та засобів розробки.

Результатом розробки системи є набір інтегрованих між собою сервісів що реалізують функціонал системи з використанням підходів предметно-орієнтованого проектування, серверних компонентів програмного

забезпечення з публічним Web-інтерфейсом на базі ASP.NET Web API та клієнтський додаток у вигляді Web-сайту, що надає доступ до серверної частини системи за допомогою браузера.

Підбиваючи підсумки, під час виконання роботи було:

- досліджено проблематику ринку автоматизації житлових приміщень;
- проведено порівняння множини рішень для автоматизації житлових приміщень;
- проведено аналіз предметної області автоматизації житлових приміщень згідно з практиками предметно-орієнтованого проектування;
- розроблено прототип системи автоматизації житлових приміщень з використанням елементів предметно-орієнтованого проектування та відповідних архітектурних практик.

Було виявлено наступні переваги застосування предметно-орієнтованого проектування для даної системи:

- реалізація логіки взаємодії з користувачем для налаштування системи зібрана у автономному обмеженому контексті, за необхідності розширення можливостей керування системою область внесення змін буде обмежена даним контекстом;
- інтеграція зі стороннім апаратним забезпеченням обмежена власним контекстом, що призводить спрощення підтримки додаткового апаратного забезпечення;
- спонукає до реалізації системи з використанням мікросервісної архітектури, що дозволяє розподілити розробку бізнес-задач на окремі команди розробки, що спеціалізуються на створенні окремих частин продукту

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Internet of Things (IoT) for building Smart Home System. // 2017 International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC) / , 2017. – (IEEE Xplore). – С. 65 - 70.
2. Технические науки в России и за рубежом: материалы IV Междунар. науч. конф. – Москва: Буки-Веди, 2015. – 140 с.
3. Internet of Things (IoT): A vision, architectural elements, and security issues. // 2017 International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC) / , 2017. – (IEEE Xplore). – С. 492 – 496.
4. Evans E. Domain-Driven Design: Tackling Complexity in the Heart of Software / Eric Evans., 2003. – 560 с
5. Вернон В. Реализация методов предметно-ориентированного проектирования / Вон Вернон. – Москва: И. Д. Вильямс, 2016. – 688 с
6. UbiquitousLanguage [Електронний ресурс]. – 2006. – Режим доступу до ресурсу: https://martinfowler.com/bliki/UbiquitousLanguage.html.
7. Вернон В. Предметно-ориентированное проектирование. Самое основное / Вон Вернон. – Чехов: Альфа-книга, 2017. – 160 с.
8. Макконнелл С. Совершенный код. Мастер-класс / Стив Макконнелл. – Русская Редакция: Вильямс, 2013. – 896 с.
9. Приемы объектно-ориентированного проектирования. Паттерны проектирования / Э.Гамма, Р. Хелм, Р. Джонсон, Д. Влиссидес. – Санкт-Петербург: Питер, 2018. – 368 с.
10. A tour of the C# language [Електронний ресурс] – Режим доступу до ресурсу: https://docs.microsoft.com/ru-ru/dotnet/csharp/tour-of-csharp/.
11. F# documentation [Електронний ресурс] – Режим доступу до ресурсу: https://docs.microsoft.com/en-us/dotnet/fsharp/.

12. .NET Core overview [Електронний ресурс] – Режим доступу до ресурсу: https://docs.microsoft.com/ru-ru/dotnet/core/about.
13. Introduction to ASP.NET Core [Електронний ресурс] – Режим доступу до ресурсу: https://docs.microsoft.com/ru-ru/aspnet/core/introduction-to-aspnet-core?view=aspnetcore-3.1.
14. Entity Framework Core Overview [Електронний ресурс] – Режим доступу до ресурсу: https://docs.microsoft.com/ru-ru/ef/core/.
15. AMQP 0-9-1 Model Explained [Електронний ресурс] – Режим доступу до ресурсу: https://www.rabbitmq.com/tutorials/amqp-concepts.html.
16. RFC 2616 - Hypertext Transfer Protocol -- HTTP/1.1 [Електронний ресурс]. – 1999. – Режим доступу до ресурсу: https://tools.ietf.org/html/rfc2616.
17. Флэнаган Д. JavaScript. Подробное руководство / Дэвид Флэнаган., 2008. – 982 с. – (5).
18. React – A JavaScript library for building user interfaces [Електронний ресурс] – Режим доступу до ресурсу: https://en.reactjs.org/.
19. Plotly JavaScript Open Source Graphing Library [Електронний ресурс] – Режим доступу до ресурсу: https://plotly.com/javascript/.
20. Docker overview [Електронний ресурс] – Режим доступу до ресурсу: https://docs.docker.com/get-started/overview/.
21. Chacon S. Pro Git / S. Chacon, B. Straub., 2020. – 527 с. – (2).
22. Beck K. Extreme Programming Explained: Embrace Change / K. Beck, A. Cynthia.. – 216 с. – (2).

ДОДАТОК А

Предметно – орієнтоване проектування системи автоматизації споруд

СПЕЦИФІКАЦІЯ

УКР.НТУУ”КПІ”.ТВ6144_20Б

Аркушів 2

Київ — 2020

Позначення	Найменування	Примітки
Документація		
УКР.НТУУ”КПІ”. ТВ6144_20Б 81-1	Дипломна записка	Зміст дипломної роботи
Компоненти		
УКР.НТУУ”КПІ”. ТВ6144_20Б 12-1	SmartHouseSystem.Devices	
УКР.НТУУ”КПІ”. ТВ6144_20Б 13-1	SmartHouseSystem.EnvironmentMonitoring	
УКР.НТУУ”КПІ”. ТВ6144_20Б 14-1	SmartHouseSystem.RoomManagement	Програмна реалізація обмеженого контексту керування акаунтами та приміщеннями
УКР.НТУУ”КПІ”. ТВ6144_20Б 15-1	SmartHouseSystem.Audit	
УКР.НТУУ”КПІ”. ТВ6144_20Б 16-1	SmartHouseSystem.Web.API	
УКР.НТУУ”КПІ”. ТВ6144_20Б 17-1	SmartHouseSystem.Web.UI	

ДОДАТОК Б

Предметно – орієнтоване проектування системи автоматизації споруд

Текст програмного модулю SmartHouseSystem.RoomManagement.Core

Аркушів 15

Київ — 2020

```
using System;

namespace RoomManagement.Core.Application
{
    public interface IDateTimeProvider
    {
        public DateTimeOffset UtcNow { get; }
    }
}


using RoomManagement.Core.Domain;
using RoomManagement.Core.Domain.Utility;
using RoomManagement.Messages.Common;
using RoomManagement.Requests;
using System.Linq;
using Direction = RoomManagement.Core.Domain.Direction;
using Influence = RoomManagement.Core.Domain.Influence;
using Period = RoomManagement.Core.Domain.Period;

namespace RoomManagement.Core.Application
{
    public static class MappingExtensions
    {
        public static RoomDto ToDto(this Room source)
        {
            var requirements = source.Requirements
                .Select(x =>
                    new RequirementDto(
                        x.Id,
                        x.DesiredValue,
                        (Parameter)x.ParameterType,
                        x.Period.ToDto()))
                .ToList();

            var actuators = source.Actuators
                .Select(x =>
                    new ActuatorDto(
                        x.Id,
                        x.Name,
                        x.Influence.ToDto()))
                .ToList();

            var sensors = source.Sensors
                .Select(x =>
                    new SensorDto(
                        x.Id,
                        x.Name,
                        (Parameter)x.TrackingParameterType))
                .ToList();

            return new RoomDto(
                source.Id,
                source.Name,
                source.Area,
                source.Created,
                requirements,
                actuators,
```

```
                sensors
            );
        }

        public static Influence ToValueObject(this Messages.Common.Influence source)
        {
            return new Influence(
                (ParameterType)source.InfluencedParamenter,
                (Direction)source.InfluenceDirection);
        }

        public static Period ToValueObject(this Messages.Common.Period source)
        {
            return new Period(source.Start, source.End);
        }
    }
}


using System;
using Shared;

namespace RoomManagement.Core.Domain
{
    public class Actuator : Entity
    {
        public string Name { get; }
        public Influence Influence { get; }

        public Actuator(Guid id, string name, Influence influence) : base(id)
        {
            Name = name;
            Influence = influence;
        }

        protected Actuator()
        {
            // for Entity Framework
        }
    }
}


namespace RoomManagement.Core.Domain
{
    public enum  Direction
    {
        Increase = 1,
        Decrease = 2
    }
}


using System.Collections.Generic;
using Shared;

namespace RoomManagement.Core.Domain
{
```

```
    public class Influence : ValueObject
    {
        public ParameterType InfluencedParameterType { get; }
        public Direction InfluenceDirection { get; }

        public Influence(ParameterType influencedParameterType, Direction
influenceDirection)
        {
            InfluencedParameterType = influencedParameterType;
            InfluenceDirection = influenceDirection;
        }

        private Influence()
        {
            // for Entity Framework
        }

        protected override IEnumerable<object> GetEqualityComponents()
        {
            yield return InfluencedParameterType;
            yield return InfluenceDirection;
        }
    }
}


namespace RoomManagement.Core.Domain
{
    public enum ParameterType
    {
        Temperature = 1,
        Humidity = 2,
        Illumination = 3
    }
}


using System;
using System.Collections.Generic;
using System.Security.Cryptography;
using System.Text;

namespace RoomManagement.Core.Domain
{
    public static class PasswordHasher
    {
        public static string Hash(string password)
        {
            var passwordBytes = Encoding.UTF8.GetBytes(password);
            var sha1 = new SHA1CryptoServiceProvider();
            var hashBytes = sha1.ComputeHash(passwordBytes);
            var hashString = Encoding.UTF8.GetString(hashBytes);
            return hashString;
        }
    }
}
```

```
using System;
using System.Collections.Generic;
using Shared;

namespace RoomManagement.Core.Domain
{
    public class Period : ValueObject
    {
        public TimeSpan Start { get; }
        public TimeSpan End { get; }

        public Period(DateTimeOffset start, DateTimeOffset end)
        {
            Start = start.TimeOfDay;
            End = end.TimeOfDay;
        }

        public Period(TimeSpan start, TimeSpan end)
        {
            Start = start;
            End = end;
        }

        private Period()
        {
            // for Entity Framework
        }

        protected override IEnumerable<object> GetEqualityComponents()
        {
            yield return Start;
            yield return End;
        }
    }
}

using System;
using Shared;

namespace RoomManagement.Core.Domain
{
    public class Requirement : Entity
    {
        public Period Period { get; }
        public ParameterType ParameterType { get; }
        public double DesiredValue { get; }
        public double Delta { get; }

        public Requirement(Guid id, Period period, ParameterType parameterType,
double desiredValue, double delta) : base(id)
        {
            Period = period;
            ParameterType = parameterType;
            DesiredValue = desiredValue;
            Delta = delta;
        }

        private Requirement()
```

```
        {
            // for Entity Framework
        }
    }
}


using System;
using System.Collections.Generic;
using System.Linq;
using RoomManagement.Core.Domain.Utility;
using RoomManagement.Events;
using Shared;

namespace RoomManagement.Core.Domain
{
    public class Room : Aggregate
    {
        public List<Sensor> Sensors { get; }
        public List<Actuator> Actuators { get; }
        public List<Requirement> Requirements { get; }

        public User User { get; private set; }

        public string Name { get; }
        public double Area { get; }

        public Room(Guid id, User user, string name, double area) : base(id)
        {
            if (area < 0)
            {
                throw new ArgumentException("Area cannot be negative");
            }

            User = user ?? throw new ArgumentNullException(nameof(user));
            Name = name ?? throw new ArgumentNullException(nameof(name));
            Area = area;
            Sensors = new List<Sensor>();
            Actuators = new List<Actuator>();
            Requirements = new List<Requirement>();
        }

        private Room()
        {
            // for Entity Framework
        }

        public void AttachSensor(string name, ParameterType parameterType)
        {
            var sensor = new Sensor(Guid.NewGuid(), name, parameterType);

            Sensors.Add(sensor);

            AddDomainEvent(
                new SensorAttached(sensor.Id, Id, name, parameterType.ToDto()));
        }

        public void AttachActuator(string name, Influence influence)
```

```
        {
            var actuator = new Actuator(Guid.NewGuid(), name, influence);

            Actuators.Add(actuator);

            var influenceDto = actuator.Influence.ToDto();
            AddDomainEvent(
                new ActuatorAttached(actuator.Id, Id, name, influenceDto));
        }

        public void AddRequirement(Period period, ParameterType parameterType, double
desiredValue, double delta = 5.0)
        {
            var requirement = new Requirement(Guid.NewGuid(), period, parameterType,
desiredValue, delta);

            Requirements.Add(requirement);

            AddDomainEvent(new RequirementCreated(
                requirement.Id,
                Id,
                requirement.Period.ToDto(),
                requirement.ParameterType.ToDto(),
                requirement.DesiredValue,
                requirement.Delta));
        }

        public void CancelRequirement(Requirement requirement)
        {
            if (Requirements.Contains(requirement))
            {
                Requirements.Remove(requirement);
                AddDomainEvent(new RequirementCancelled(Id, requirement.Id));
            }
        }
    }
}


using System;
using Shared;

namespace RoomManagement.Core.Domain
{
    public class Sensor : Entity
    {
        public string Name { get; }
        public ParameterType TrackingParameterType { get; }

        public Sensor(Guid id, string name, ParameterType trackingParameterType) :
base(id)
        {
            Name = name;
            TrackingParameterType = trackingParameterType;
        }

        public Sensor()
        {
```

```
            // for Entity Framework
        }
    }
}


using System;
using System.Collections.Generic;
using RoomManagement.Events;
using Shared;

namespace RoomManagement.Core.Domain
{
    public class User : Aggregate
    {
        public string Login { get; }

        public string PasswordHash { get; private set; }

        public List<Room> Rooms { get; set; }

        public User(Guid id, string login, string password) : base(id)
        {
            Login = login;
            PasswordHash = PasswordHasher.Hash(password);
            Rooms = new List<Room>();
        }

        public User()
        {
            // for Entity Framework
        }

        public void AddRoom(Room room)
        {
            Rooms.Add(room);
            AddDomainEvent(new RoomCreated(Id, room.Id, room.Name, room.Area));
        }

        public bool PasswordMatches(string password)
        {
            var hashedPassword = PasswordHasher.Hash(password);
            return hashedPassword == PasswordHash;
        }

        public void ResetPassword(string newPassword, string oldPassword)
        {
            if (PasswordMatches(oldPassword))
            {
                var newPasswordHash = PasswordHasher.Hash(newPassword);
                PasswordHash = newPasswordHash;
            }
        }
    }
}


using MediatR;
```

```
using RoomManagement.Core.Domain;
using RoomManagement.Core.Domain.Interfaces;
using RoomManagement.Requests;
using Shared;
using System.Collections.Generic;
using System.Linq;
using System.Threading;
using System.Threading.Tasks;

namespace RoomManagement.Core.Application.CommandHandlers
{
    public class RoomHandler :
        IRequestHandler<CreateRoom, Result>,
        IRequestHandler<AttachSensor, Result>,
        IRequestHandler<AttachActuator, Result>,
        IRequestHandler<CreateRequirement, Result>,
        IRequestHandler<CancelRequirement, Result>,
        IRequestHandler<GetRooms, Result<IReadOnlyList<RoomListItemDto>>>,
        IRequestHandler<GetRoom, Result<RoomDto>>
    {
        private readonly IRoomRepository _roomRepository;
        private readonly IUserRepository _userRepository;

        public RoomHandler(
            IRoomRepository roomRepository,
            IUserRepository userRepository)
        {
            _roomRepository = roomRepository;
            _userRepository = userRepository;
        }

        public async Task<Result> Handle(CreateRoom request, CancellationToken
cancellationToken)
        {
            var userResult = await _userRepository.GetByIdAsync(request.UserId);
            if (userResult.IsSuccess)
            {
                var user = userResult.Value;
                var room = new Room(request.RoomId, user, request.Name,
request.Area);

                await _roomRepository.CreateAsync(room);

                await
_roomRepository.UnitOfWork.SaveEntitiesAsync(cancellationToken);
                return Result.Ok();
            }
            return Result.Fail("User not found for current room.");
        }

        public async Task<Result> Handle(AttachSensor request, CancellationToken
cancellationToken)
        {
            var roomResult = await _roomRepository.GetByIdAsync(request.RoomId);
            if (roomResult.IsSuccess)
            {
                var room = roomResult.Value;
```

```
                room.AttachSensor(
                    request.Name,
                    (ParameterType)request.TrackingParameter);

                await
_roomRepository.UnitOfWork.SaveEntitiesAsync(cancellationToken);
                return Result.Ok();
            }
            return Result.Fail("Room not found.");
        }

        public async Task<Result> Handle(AttachActuator request, CancellationToken
cancellationToken)
        {
            var roomResult = await _roomRepository.GetByIdAsync(request.RoomId);
            if (roomResult.IsSuccess)
            {
                var room = roomResult.Value;

                room.AttachActuator(
                    request.Name,
                    request.Influence.ToValueObject());

                await
_roomRepository.UnitOfWork.SaveEntitiesAsync(cancellationToken);
                return Result.Ok();
            }
            return Result.Fail("Room not found.");
        }

        public async Task<Result> Handle(CreateRequirement request, CancellationToken
cancellationToken)
        {
            var roomResult = await _roomRepository.GetByIdAsync(request.RoomId);
            if (roomResult.IsSuccess)
            {
                var room = roomResult.Value;

                room.AddRequirement(
                    request.Period.ToValueObject(),
                    (ParameterType)request.Parameter,
                    request.DesiredValue,
                    request.Delta);

                await
_roomRepository.UnitOfWork.SaveEntitiesAsync(cancellationToken);
                return Result.Ok();
            }
            return Result.Fail("Room not found.");
        }

        public async Task<Result> Handle(CancelRequirement request, CancellationToken
cancellationToken)
        {
            var roomResult = await _roomRepository.GetByIdAsync(request.RoomId);
            if (roomResult.IsSuccess)
            {
                var room = roomResult.Value;
```

```
                var requirement = room.Requirements.SingleOrDefault(x => x.Id ==
request.RequirementId);

                if (requirement != null)
                {
                    room.CancelRequirement(requirement);

                    await
_roomRepository.UnitOfWork.SaveEntitiesAsync(cancellationToken);
                    return Result.Ok();
                }
                else
                {
                    return Result.Fail("Requirement not found for given room.");
                }
            }
            return Result.Fail("Room not found.");
        }

        public async Task<Result<IReadOnlyList<RoomListItemDto>>> Handle(GetRooms
request, CancellationToken cancellationToken)
        {
            var rooms = await _roomRepository.GetByUserAsync(request.UserId);

            var roomDtos = rooms
                .Select(x => new RoomListItemDto(x.Id, x.Name, x.Area, x.Created))
                .ToList();

            return Result.Ok<IReadOnlyList<RoomListItemDto>>(roomDtos);
        }

        public async Task<Result<RoomDto>> Handle(GetRoom request, CancellationToken
cancellationToken)
        {
            var roomResult = await _roomRepository.GetByIdAsync(request.RoomId);
            if (roomResult.IsSuccess)
            {
                var room = roomResult.Value;
                var roomDto = room.ToDto();
                return Result.Ok(roomDto);
            }
            return Result.Fail<RoomDto>("Room not found.");
        }
    }
}


using MediatR;
using RoomManagement.Core.Domain;
using RoomManagement.Core.Domain.Interfaces;
using RoomManagement.Requests;
using Shared;
using System.Linq;
using System.Threading;
using System.Threading.Tasks;

namespace RoomManagement.Core.Application.CommandHandlers
{
```

```
    public class UserHandler :
        IRequestHandler<CreateUser, Result>,
        IRequestHandler<ChangePassword, Result>,
        IRequestHandler<GetUser, Result<UserGeneralInfoDto>>,
        IRequestHandler<AuthenticateUser, Result<AuthResultDto>>
    {
        private readonly IUserRepository _userRepository;

        public UserHandler(IUserRepository userRepository)
        {
            _userRepository = userRepository;
        }

        public async Task<Result> Handle(CreateUser request, CancellationToken
cancellationToken)
        {
            var userWithSameLoginExists = await
_userRepository.ExistsWithLoginAsync(request.Login);
            if (userWithSameLoginExists)
            {
                return Result.Fail("User with same login already exists.");
            }

            var user = new User(request.UserId, request.Login, request.PasswordHash);
            await _userRepository.CreateUserAsync(user);
            await _userRepository.UnitOfWork.SaveEntitiesAsync(cancellationToken);

            return Result.Ok();
        }

        public async Task<Result> Handle(ChangePassword request, CancellationToken
cancellationToken)
        {
            var userResult = await _userRepository.GetByIdAsync(request.UserId);
            if (userResult.IsSuccess)
            {
                var user = userResult.Value;

                if (user.PasswordMatches(request.OldPassword))
                {
                    user.ResetPassword(request.NewPassword, request.OldPassword);
                    await
_userRepository.UnitOfWork.SaveEntitiesAsync(cancellationToken);
                    return Result.Ok();
                }
                else
                {
                    return Result.Fail("Incorrect old password.");
                }
            }
            else
            {
                return Result.Fail("User not found");
            }
        }

        public async Task<Result<UserGeneralInfoDto>> Handle(GetUser request,
CancellationToken cancellationToken)
```

```
        {
            var userResult = await _userRepository.GetByIdAsync(request.UserId);
            if (userResult.IsSuccess)
            {
                var user = userResult.Value;
                var roomsIds = user.Rooms.Select(x => x.Id).ToArray();
                var devicesTotal = await
_userRepository.GetTotalNumberOfDevicesAsync(user.Id);
                var userDto = new UserGeneralInfoDto(user.Id, user.Login, roomsIds,
devicesTotal);
                return Result.Ok(userDto);
            }
            else
            {
                return Result.Fail<UserGeneralInfoDto>("User not found.");
            }
        }

        public async Task<Result<AuthResultDto>> Handle(AuthenticateUser request,
CancellationToken cancellationToken)
        {
            var userResult = await _userRepository.GetByLoginAsync(request.Login);
            if (userResult.IsSuccess)
            {
                var user = userResult.Value;
                var canAuthenticate = user.PasswordMatches(request.Password);
                if (canAuthenticate)
                {
                    return Result.Ok(new AuthResultDto(user.Id, user.Login));
                }

                return Result.Fail<AuthResultDto>("Login or password is incorrect");
            }
            return Result.Fail<AuthResultDto>("Login or password is incorrect");
        }
    }
}


using System;
using System.Collections.Generic;
using System.Text;
using System.Threading.Tasks;
using Shared;

namespace RoomManagement.Core.Domain.Interfaces
{
    public interface IRepository<TAggregate> where TAggregate : Aggregate
    {
        IUnitOfWork UnitOfWork { get; }

        Task<Result<TAggregate>> GetByIdAsync(Guid id);
    }
}


using System;
using System.Collections.Generic;
```

```
using System.Threading.Tasks;

namespace RoomManagement.Core.Domain.Interfaces
{
    public interface IRoomRepository : IRepository<Room>
    {
        Task<IList<Room>> GetByUserAsync(Guid userId);

        Task CreateAsync(Room newRoom);

    }
}


using System.Threading;
using System.Threading.Tasks;

namespace RoomManagement.Core.Domain.Interfaces
{
    public interface IUnitOfWork
    {
        Task SaveEntitiesAsync(CancellationToken cancellationToken = default);
    }
}


using System;
using System.Threading.Tasks;
using Shared;

namespace RoomManagement.Core.Domain.Interfaces
{
    public interface IUserRepository : IRepository<User>
    {
        Task CreateUserAsync(User newUser);
        Task<bool> ExistsWithLoginAsync(string login);
        Task<int> GetTotalNumberOfDevicesAsync(Guid userId);
        Task<Result<User>> GetByLoginAsync(string login);
    }
}


namespace RoomManagement.Core.Domain.Utility
{
    public static class MappingExtensions
    {
        public static Messages.Common.Parameter ToDto(this ParameterType source)
        {
            return (Messages.Common.Parameter) source;
        }

        public static Messages.Common.Direction ToDto(this Direction source)
        {
            return (Messages.Common.Direction)source;
        }
        public static Messages.Common.Period ToDto(this Period source)
        {
            return new Messages.Common.Period(source.Start, source.End);
```

```
        }

        public static Messages.Common.Influence ToDto(this Influence source)
        {
            var destination = new Messages.Common.Influence(
                source.InfluencedParameterType.ToDto(),
                source.InfluenceDirection.ToDto());

            return destination;
        }
    }
}
```

ДОДАТОК В

Предметно – орієнтоване проектування системи автоматизації споруд

Опис програмного компоненту SmartHouseSystem.RoomManagement

Аркушів 9

Київ — 2020

АНОТАЦІЯ

Компонент SmartHouseSystem.RoomManagement являє собою програмну реалізацію обмеженого контексту керування акаунтом користувача та конфігурації приміщень. Компонент містить модель предметної області, що складається з класів сутностей, класів об’єктів-значень тощо та набір обробників команд і запитів що призначені для реалізації окремих прецедентів обмеженого контексту.

Даний модуль виконує наступні задачі:

— керування акаунтами користувачів;
— керування приміщеннями користувачів;
— реалізація окремих прецедентів системи стосовно контексту керування акаунтом користувача та конфігурації приміщень;
— ініціювання подій предметної області контексту керування акаунтом користувача та конфігурації приміщень.

Вхідними даними являються команди та запити, що представляють собою контракти обмеженого контексту із іншими компонентами. Вихідними даними є результати виконання запитів та команд, множина ініційованих інтеграційних подій для розповсюдження змін стану системи до інших обмежених контекстів.

Розробка даного модулю системи була виконана в інтегрованому середовищі розробки Visual Studio 2019, мовами програмування C# та F#. Цільова програмна платформа даного модулю — NET Core 3.1.

ЗМІСТ

ЗАГАЛЬНІ ВІДОМОСТІ

Програмний модуль SmartHouseSystem.RoomManagement являє собою програмну реалізацію обмеженого контексту керування акаунтом користувача та конфігурації приміщень. До складу модулю входить модель предметної області обмеженого контексту, веб-інтерфейс для взаємодії із компонентом за допомого протоколу HTTP, інфраструктурні коди для налагодження інтеграції модуля з базою даних та брокером повідомлень.

Більша частина модулю написана мовою програмування C# 8.0, для програмної платформи .NET Core 3.1. Компонент SmartHouseSystem.RoomManagement.Messages написаний мовою F# 4.6.

ФУНКЦІОНАЛЬНЕ ПРИЗНАЧЕННЯ

Модуль SmartHouseSystem.RoomManagement призначений для реалізації прецедентів системи автоматизації житлових приміщень, що стосуються авторизації, створення та керування акаунтом, створення та налаштування житлових приміщень користувача.

До функціональних можливостей модуля відносяться:

— автентифікація користувачів;
— створення акаунтів користувачів;
— налаштування акаунтів користувачів;
— створення приміщень;
— додавання вимірювальних та регулюючих приладів до приміщень.

ОПИС ЛОГІЧНОЇ СТРУКТУРИ

Структурно модуль складається з чотирьох компонентів:

— SmartHouseSystem.RoomManagement.Messages містить набір типів-повідомлень що є контрактами обмеженого контексту з іншими модулями системи. До повідомлень належать команди, запити та події;

— SmartHouseSystem.RoomManagement.Core містить модель предметної області обмеженого контексту та набір обробників команд, запитів та подій. Компонент визначає необхідні для його функціонування абстракції та не залежить від жодних інфраструктурних елементів системи;

— SmartHouseSystem.RoomManagement.Infrastructure реалізує визначені у Core компоненті абстракції та виконує інфраструктурні потреби модулю, до котрих відноситься доступ до постійного сховища даних, інтеграція з брокером повідомлень тощо;

— SmartHouseSystem.RoomManagement.API є веб-інтерфейсом обмеженого контексту керування акаунтом користувача та конфігурації приміщень, до його обов’язків відноситься обробка HTTP запитів, схема автентифікації тощо.

ВИКОРИСТОВУВАНІ ТЕХНІЧНІ ЗАСОБИ

Компонент SmartHouseSystem.RoomManagement розроблено за домопогою програмної платформи .NET Core 3.1. Для забезпечення розробки веб-інтерфейса модуля було використано технології ASP.NET Core 3.1. Інфраструктурні коди доступу до бази даних використовують технологію Entity Framework Core 3.1. В якості механізму надсилання та обробки команд, запитів та подій було використано бібліотеку MediatR 8.0.1.

Мовою програмування для написання модуля біуло обрано C# 8.0. Для пришвидшення розробки було прийнято рішення обрати в якості мови програмування F# 4.6 для опису типів-повідомлень у компоненті SmartHouseSystem.RoomManagement.Messages.

ВИКЛИК І ЗАВАНТАЖЕННЯ

Модуль SmartHouseSystem.RoomManagement є частиною серверного програмного забезпечення системи автоматизації житлових приміщень та може бути запущений за допомогою dotnet-cli, або ж за допомогою системи управління контейнерами Docker.

ВХІДНІ ТА ВИХІДНІ ДАНІ

Модуль SmartHouseSystem.RoomManagement представлений у вигляді веб-інтерфесу, отже вхідні та вихідні дані мають вигляд HTTP запитів та відповідей. Компонент SmartHouseSystem.RoomManagement.API виступає у ролі посередника між основною логікою програмного модуля та протоколом HTTP. Після обробки прийнятого запиту, модуль використовує типи-повідомлення, до яких належать команди та запити.

Вхідні дані — це насамперед дані, введені користувачем під час роботи з інтерфейсною частиною системи. Вихідні дані — результат обробки певних даних користувача та виведення існуючих даних за запитом.

ВІДГУК

керівника дипломної роботи

освітньо-кваліфікаційного рівня „бакалавр”

виконаної на тему : “Предметно – орієнтоване проектування системи автоматизації споруд”

студентом Ташу Альбертом Аркадійовичем

(прізвище, ім’я, по батькові)

(складається у довільній формі із зазначенням: головної цілі дипломної роботи, в інтересах або на замовлення якої організації він виконаний (в рамках науково дослідної роботи кафедри, підприємства, НДІ тощо); відповідності виконаної ДР завданню; ступеня самостійності при виконанні ДР; рівня підготовленості дипломника до прийняття сучасних рішень; уміння аналізувати необхідні літературні джерела, приймати правильні(інженерні, наукові) рішення, застосовувати сучасні системні та інформаційні технології, проводити фізичне або математичне моделювання, обробляти та аналізувати результати експерименту; найбільш важливих теоретичних та практичних результатів апробації їх(участь у конференціях, семінарах, оформлення патентів, публікація в наукових журналах тощо); загальної оцінки виконаної ДР, відповідності якості підготовки дипломника вимогам ОКХ і можливості присвоєння йому відповідної кваліфікації; інші питання, які характеризують професійні якості дипломника)

Керівник дипломної роботи

доцент, канд. ф.-м. наук _____ Тарнавський Ю. А.

(посада, вчені звання, ступінь) (підпис) (ініціали, прізвище)

РЕЦЕНЗІЯ

на здобуття ступеня бакалавра,
виконаний на тему: “ Предметно – орієнтоване проектування системи автоматизації споруд ”
студентом Ташу Альбертом Аркадійовичем

(складається у довільній формі із зазначенням: відповідності ДР затвердженій темі та завданню на дипломне проектування; актуальності теми; реальності ДР(його виконання на замовлення підприємств, організацій, за науковою тематикою кафедри, НДІ тощо); глибину техніко-економічного обґрунтування прийняття рішень; ступеня використання сучасних досягнень науки, техніки, виробництва, інформаційних та інженерних технологій; оригінальності прийнятих рішень та отримання результатів; правильності проведених розрахунків і конструкторсько-технологічних рішень; наявності і повноти експериментального (фізичного або математичного) підтвердження прийнятих рішень; якості виконання пояснювальної записки, відповідності креслень вимогам ДСТУ, ЕСКД; можливості впровадження результатів ДР; недоліків ДР; оцінки ДР за 4-бальною системою і можливості присвоєння дипломнику відповідної кваліфікації (формулювання згідно з навчальним планом напряму підготовки або спеціальності).

Рецензент

_____	_____	_____
(посада, вчені звання, ступінь)	(підпис)	(ініціали, прізвище)